Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

A Post-Attack Recovery Architecture for Smart Electricity Meters

Eine Architektur zur Kontrollwiederherstellung nach Angriffen auf Smart Metering in Stromnetzen

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von: Jan Sebastian Götte
geboren am: Aus Datenschutzgründen nicht abgedruckt
geboren in: Aus Datenschutzgründen nicht abgedruckt

Gutachter/innen: Prof. Dr. Björn Scheuermann
Prof. Dr.-Ing. Eckhard Grass

eingereicht am: ..... verteidigt am: .....

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 31.03.2020 .....

Contents

Chapter 1

Introduction

Like in all fields of engineering there is an ongoing diffusion of information systems into industrial control systems in the power grid. Automation of these control systems has been practised for the better part of a century already. Until recently this automation was mostly limited to core components of the grid. Generators in power stations are computer-controlled according to electromechanical and economic models. Switching in substations is automated to allow for fast failure recovery. Humans are still vital to these systems, but their tasks have shifted from pure operation to engineering, maintenance and surveillance.

A large-scale trend in power systems is the move from a model of centralized generation built around massive large-scale fossil and nuclear power plants towards a more heterogenous model. In this new model large-scale fossil power plants still serve a major role but two new factors come into play. One is the advance of renewable energies. The large-scale use of wind and solar power in particular from a current standpoint seems unavoidable for our continued existence on this planet. For the electrical grid however, these systems constitute a significant challenge. Fossil-fueled power plants can be precisely controlled to match the expected energy consumption at any point in time. This tracking of production and consumption is vital to the stability of the grid. Renewable energies such as wind and solar power do not provide the same degree of controllability, and they introduce a large degree of uncertainty due to the unpredictable way of the forces of nature.

Along with this change in dynamic behavior renewable energies have brought forth the advance of distributed generation. In distributed generation end-customers that previously only consumed energy have started to feed energy into the grid from small solar installations on their property. Distributed generation is a chance for customers to gain autonomy and shift from a purely passive role to being active participants of the electricity market[46].

To match this new landscape of decentralized generation and unpredictable renewable resources the utility industry has had to adapt itself in major ways. One aspect of this adaption that is particularly visible to ordinary people is the computerization of end-user energy metering. Despite the widespread use of industrial control systems inside the electrical grid and the far-reaching diffusion of computers into people's everyday lifes the energy meter has long been one of the last remnants of an offline, analog time. Until the 2010s many of the world's households were still served through electromechanical Ferraris-style meters that have their origin in the late 19th century.

Today under the umbrella term *Smart Grid* the shift towards fully computerized, often networked meters has been partially accomplished. The roll out of these *Smart Meters* has not been very smooth overall with some countries severely lagging behind other countries. As a

safety-critical technology smart meter technology is usually standardized on a per-country basis. This leads to an inhomogenous landscape with in some instances wildly incompatible systems. Often vendors only serve a single country or have a separate model of their meter for each country. This complex standardization landscape and market situation has led to a proliferation of highly complex, custom-coded microcontroller firwmare. The complexity and scale of this often network-connected firmware makes for a ripe substrate for bugs to surface.

A remotely exploitable flaw inside a smart meter's firmware[1] could have consequences ranging from impaired billing functionality to an existential threat to grid stability. A coördinated attack on meters in a country where load switches are common could at worst cause widespread activation of grid safety systems by repeatedly connecting and disconnecting megawatts of load capacity in just the wrong moments.

Mitigation of these attacks through firmware security measures is unlikely to yield satisfactory results. The enormous complexity of smart meter firmware makes firmware security extremely labor-intensive. The diverse standardization landscape makes a coördinated, comprehensive response unlikely.

In this thesis instead of lamenting the state of firmware security we introduce a pragmatic solution to the in our minds likely scenario of a large-scale compromise of smart meter firmware. In our proposal the components of the smart meter that are threatened by remote compromise are equipped with a physically separate *safety reset controller* that listens for a reset command transmitted through the electrical grid itself and on reception forcibly resets the smart meter's entire firmware to a known-good state. Our safety reset controller receives commands through Direct Sequence Spread Spectrum (DSSS) modulation carried out on grid frequency through a large controllable load such as an aluminium smelter. After forward error correction and cryptographic verification it re-flashes the target application microcontroller over the standard JTAG interface.

In this thesis starting from a high-level architecture we have carried out extensive simulations of our proposal's performance under real-world conditions. Based on these simulations we implemented an end-to-end prototype of our proposed safety reset controller as part of a realistic smart meter demonstrator. Finally we experimentally validate our results and give an outline of further steps towards practical implementation.

1.1 Structure and operation of the electrical grid

Since this thesis is filed under *computer science* we will provide a very brief overview of some basic aspects of modern power grids.

[1] There are several smart metering architectures that ascribe different roles to the component called *smart meter*. Coarsely divided into two camps these are systems where all metering and communication code resides within one physical unit and systems where metering and communication are separated into two units, the *smart meter* and the *smart meter gateway*. An example for the former are setups in the USA, an example of the latter is the one in Germany. For clarity in this introductory chapter we use *smart meter* to describe the entire system at the customer premises including both the meter and a potential gateway.

1.1.1 Structure of the electrical grid

The electical grid is composed of a large number of systems such as distribution systems, power stations and substations interconnected by long transmission lines. Mostly due to ohmic losses[2] the efficiency of transmission of electricity through long transmission lines increases with the square of voltage[45, 42]. In practice economic considerations take into account a reduction of the considerable transmission losses (about 6 % in case of Germany[52]) as well as the cost of equipment such as additional transformers and the cost increase for the increased volatage rating of components such as transmission lines. Overall these considerations have led to a hierarchical structure where large amounts of energy are transmitted over very long distances (up to thousands of kilometers) at very high voltages (upwards of 200 kV) and voltages get lower the closer one gets to end-customer premises. In Germany at the local level a substation will distribute 10 kV to 30 kV to large industrial consumers and streets with small transformer substations converting this to the 400 V three-phase AC households are usually hooked up with[45].

Transmission lines, bus bars and tie lines

The number one component of the electrical grid are transmission lines. Short transmission lines that tightly couple parts of a substation are called *bus bars*. Transmission lines that couple otherwise independent grid segments are called *tie lines*. A tie line often connects grid segments operated by two different operators e.g. across a country border.

Short transmission lines can be approximated as a simple lumped-component RLC[3] circuit. In this case the effect of wave propagation along the line does not have to be taken into consideration. In this lumped model the transmission line is represented by a circuit of one or two inductors, one or two capacitors and some resistors. This representation simplifies analysis. For *long* transmission lines above 50 km (cable) or 250 km (overhead lines) this approximation breaks down and wave propagation along the line's length has to be taken into account. The resulting model is what RF engineering calls a *transmission line* and models the line's parasitics[4] as being uniformly distributed along the length of the line. To approximate this model in lumped-element evaluations the line is represented as a long chain of small lumped-component RLC sections. This complex structure makes modelling more difficult in comparison to short lines[45].

Almost all transmission lines used in the transmission and distribution grid use three-phase AC. Long-distance overland lines are usually implemented as overhead lines due to their low cost and ease of maintenance. Underground cables are much more expensive due to their isolation and are only used when overhead lines cannot be used for e.g. safety or aesthetic reasons. In some specialized applications such as long, high-power undersea cables high-voltage DC (HVDC) is used. In HVDC converter stations at both ends of the line convert between three-phase AC and the line's DC voltage. These converter stations are controlled electronically and do not exhibit any of the electromechanical effects generators in a power plant do. Since HVDC

[2]Power dissipation of a resistor of resistance $R[\Omega]$ given current $I[A]$ is $P_{\text{loss}}[W] = U_{\text{drop}} \cdot I = I^2 \cdot R$. Fixing power $P_{\text{transmitted}}[W] = U_{\text{line}} \cdot I$ this yields a dependency on line voltage $U_{\text{line}}[V]$ of $P_{\text{loss}} = \left(\frac{P_{\text{transmitted}}}{U_{\text{line}}}\right)^2 \cdot R$. Thus, ignoring other losses a $2\times$ increase in transmission voltage halves current and cuts ohmic losses to a quarter. In practice the economics of this are much more complicated due to the cost of better isolation for higher-voltage parts and the added factor of power factor compensation.

[3]resistor-inductor-capacitor

[4]stray capacitance, ohmic resistance and stray inductance

re-synthesizes three-phase AC from DC at the receiving end of the line it can be used to couple non-synchronous grids. This also allows for additional degrees of control over the transmission of power compared to a regular transmission line. These technical benefits are offset by the high initial cost (mostly due to the converter stations) leading to HVDC being used in specific situations only[46].

Generators

Traditionally all generators in the power grid were synchronous machines. A synchronous machine is a generator that is wound and connected in such a way that during normal operation its rotation is synchonous with the grid frequency. Grid frequency and generator rotation speed are bidirectionally electromechanically coupled. If a generator would lag behind the grid it would receive electrical energy from the grid and convert it into mechanical energy, acting as a motor. Small deviations between rotational speed and grid frequency will be absorbed by the electromechanical coupling between both. All generators connected to the grid operate synchronously. Maintaining this synchronization over time is the task of complex control systems within each power station[42, 45].

Nowadays besides traditional rotating generators the grid also contains a large amount of electronically controlled inverters. These inverters are used in photovoltaic installations and other setups where either DC or non-synchronous AC is to be fed into the grid. Setups like this behave differently to rotating generators. In particular *inertia* in these setups is either absent or a software parameter potentially reducing their overload capacity compared to rotating generators. The fundamentally different nature of electronically controlled inverters has to be taken into account in planning and regulation[46].

Switchgear

In the electrical grid switches perform various roles. The ones a computer scientist would recognize are used for routing electricity between transmission lines and transformers and can be classified into ones that can be switched under load (called load switches) and ones that can not (called disconnectors). The latter are used to ensure parts of the network are free from voltage. The former are used to re-route flows of electrical currents. A major difference in their construction is that in contrast to disconnectors load switches have built-in components that extinguish the high-power arc discharge that forms when the circuit is interrupted under load[5]. Beyond this there are circuit breakers. Circuit breakers are safety devices that can still switch even under failure conditions at several times the circuit's nominal current. They are activated automatically on conditions such as overcurrent or overvoltage. Fuses can be considered non-resettable switches. The fuse in a computer power supply is barely more than a glass tube with some wire in it that is designed to melt at the designated current. In energy systems fuses are often much more complex devices that in some cases even utilize explosivese to quickly and decisively open the circuit and extinguish the resulting arc discharge[104, 45, 42].

Transformers

Along with transmission lines transformers are one of the main components most people will be thinking of when talking about the electrical grid. Transformers connect grid segments at

[5]While an arc discharge is considered a fault condition in most low-voltage systems including computers, in energy systems it is often part of normal operation.

different voltage levels with one another. In the distribution grid transformers are used to provide standard end-user voltage levels to the customer (e.g. 230/400V in Europe) from a 10 kV to 25 kV feeder. Transformers can also be used to convert between buses without a fourth neutral conductor and buses with one.

Transformers are large and heavy devices consisting of thick copper wire or copper foil windings arranged around a core made from thin stacked, insulated iron sheets. The entire core sits within a large metal enclosure that is filled with liquid (usually a specialized oil) for both cooling and electrical insulation. This cooling liquid is cooled by means such as radiator fins on the transformer enclosure itself or an external radiator. Depending on the design cooling may rely on natural convection within the cooling liquid or on electrical pumps[45, 42].

Transformers come in a large variety of coil and wiring configurations. There exist autotransformers where the secondary is part of the primary (or vice-versa) that are used to translate between voltage levels without galvanic isolation at lower cost. Transformers used in parts of the electrical grid often have several taps and include *tap changers*. A tap changer is a system of mechanical switches that can be used to switch between several discrete transformer ratios to adjust secondary voltage under load[42]. Tap changers are used in the distribution grid to maintain the specified voltage tolerances at the customer's connection.

Instrument transformers

While operating on the exact same physical principles instrument transformers are very different from regular transformers in an energy system. Instrument transformers are specialized low-power transformers that are used as transducers to measure voltage or current at very high voltages. They are part of the control and protection systems of substations[45].

Chokes

Chokes are large inductors. In power grid applications their construction is similar to the construction of a transformer with the exception that they only have a single winding on the core. They are used for a variety of purposes. A frequent use is as a series inductor on one of the phases or the neutral connection to limit transient fault currents. In addition to use as simple series inductances for current limiting inductors are also used to tune LC circuits. One such use are Petersen coils, large inductors in series with the earth connection at a transformer's star point are used to quickly extinguish arcs between phase and ground on a transmission line. The Petersen coil forms a parrallel LC resonant circuit with the transmission line's earth capacitance. Tuning this circuit through adjusting the petersen coil reduces earth fault current to levels low enough to quickly extinguish the arc[42].

Power factor correction

Reactive power (also referred to as VAR after its is unit Volt-Ampère Reactive) an important variable in the operation of electrical grids (see sec. 3.1.1). If reactive power generation and consumption are mismatched, high currents develop that lead to high transmission losses. For this reason grids include circuits to compensate reactive power imbalances[45]. These circuits can be as simple as inductors or capacitors connected to a power line but often can be switched to adapt to changing load conditions. Static Var compensators are particularly fast-acting reactive power compensation devices whose purpose is to maintain bus voltage[111].

Loads

Lastly, there is the loads that the electrical grid serves. Loads range from mains-powered indicator lights in devices such as light switches or power strips weighing in at mere milliwatts to large smelters in industrial metal production that can consume a good fraction of a gigawatt all on their own.

1.1.2 Operational concerns

Modelling the electrical grid

Modelling performs an important role in the engineering of a reliable power infrastructure. The grid is a complex, highly dynamic system. To maintain operational parameters such as voltage in various parts of the grid, grid frequency and currents inside their specified ranges complex control systems are necessary. To design and parametrize such control systems simulations are a valuable tool. Using model calculations the effects of control systems on operational variables such as transmission efficiency or generation losses can be estimated. Model simulations can be used to identify structural issues such as potential points of congestion. The same models can then be used to engineer solutions to such issues, e.g. by simulating the effect of a new transmission line.

There are several aspects under which the grid or parts of the grid can be simulated. There are static analysis methods such as modal analysis that yield information on electromechanical oscillations by computing the eigenvalues of a large system of differential equations describing the collective behavior of all components of the grid. Modal analysis is one example of simulations used in grid planning. Using modal analysis likely oscillatory modes can be identified and ultimately these results can inform a decision to install additional stabilization systems in a particular location. In contrast to static analysis, transient simulations calculate an approximation of the time-domain behavior of some variable of interest under a given model. Transient simulations are used e.g. in the design of control systems. Power flow equations describe the flow of electrical energy throughout the network from generator to load. Numerical solutions these equations are used to optimize control parameters to increase overall efficiency.

1.2 Smart meter technology

Smart meters were a concept pushed by utility companies throughout the 00's. Smart metering is one component of the larger societal shift towards digitally interconnected technology. Old analog meters required that service pesonnel physically come to read the meter. *Smart* meters automatically transmit their readings through modern technologies. Utility companies were very interested in this move not only because of the cost savings for meter reading personnel. Beyond this, an always-connected meter allows several entirely new use cases that have not been possible before. One often-cited one is utilizing the new high-resolution load data to improve load forecasting to allow for greater generation efficiency. Computerizing the meter also allows for new fee models where electricity cost is no longer fixed over time but adapts to market conditions. Models such as prepayment electricity plans where the customer is automatically disconnected until they pay their bill are significantly aided by a fully electronic system that can be controlled and monitored remotely. A remotely controllable load switch can also be used

to coerce customers in situations where that was not previously economically possible[6].

To the customer the utility of a smart meter is largely limited to the convenience of being able to read it without going to the basement. In the long term it is said that there will be second-order savings to the customer since electricity prices adapting to the market situation along with this convenience will lead them to consume less electricity and to consume it in a way that is more amenable to utilities, both leading to reduced cost.

Traditional Ferraris counters with their distinctive rotating aluminium disc are simple electromechanical devices. Since it does not include any failure-prone semiconductors or other high technology a cheap Ferraris-style meter can easily last decades. In contrast to this, smart meters are complex high technology. They are vastly more expensive to develop in the first place since they require the development and integration of large amounts of complex, custom firwmare. Once deployed, their lifetime is severely limited by this very complexity. Complex semiconductor devices tend to fail, and firmware that needs to communicate with the outside world tends to not age well. This combination of higher unit cost and lower expected lifetime leads to grossly increased costs per household. This cost is usually shared between utility and customer.

As part of its smart metering rollout the German government in 2013 had a study conducted on the economies of smart meter installations. This study came to the conclusion that for the majority of households computerizing an existing ferraris meter is uneconomical. For larger consumers or new installations the higher cost of installation over time is offset by the resulting savings in electricity cost[38].

1.2.1 Human-Computer Interaction aspects of smart meter technology

A fundamental aspect in realizing the cost and energy savings promised by the smart metering revolution is that it requires a paradigm shift in consumer interaction. Previously most consumers would only confront their energy use when their monthly or yearly electricity bill arrived. All of the cost savings smart meters promise over traditional metering infrastructure[7] critically depend on the consumer regularly interacting with the meter through an in-home display or app. We live in an era where our attention is already highly contested. A myriad of apps and platforms compete for our attention through our smart phones and other devices. Introducing an entirely new service into this already complex battleground is a large endeavour. On the one hand it is not clear how this new service would compete with everything else. On the other hand if it does manage to capture our attention and lead us to modify our behavior, what are the side effects? For instance, does an in-home display increase financial anxiety in economically disadvantaged customers?

Human Computer Interaction research has touched the topic of smart metering several times and has many insights to offer for technologists[109, 110, 95, 43, 65]. An issue pointed out in Rodden et al. [110] is that at least in some countries consumers fundamentally distrust their utility companies. This trust issue is exacerbated by smart meters being unilaterally forced onto

[6]The swiss association of electrical utility companies in sec. 7.2 par. (2)a of their 2010 whitepaper on the introduction of smart metering[13] cynically writes that remotely controllable load switches "lead a new tenant to swiftly register" with the utility company. This whitepaper completely vanished from their website some time after publication, but the internet archive has a copy.

[7]We are excluding savings from Demand-Side Response (DSR) implemented through smart meters here: Traditional ripple control systems already allowed for these, and due to the added cost of high-power relays many smart meters do not include such features.

consumers by utility companies. Much of the success of smart metering's ubiquitous promises of energy savings fundamentally depends on consumer coöperation. Here, the aforementioned trust issue calls into question smart metering's chances of long-term success.

As pierce01 pointed out smart metering developments could benefit greatly from early involvement of HCI research. HCI research certainly would not have overlooked entire central issues such as privacy as it happened in the dutch case[47]. The current corporate-driven approach to a technological advance forced through national standardization bears a serious risk of failing to meet its ostensible objectives for consumers. The role of consumers and the complex sociotechnological environment posed by this new technology is seriously considered nowhere in the standardization process. While certainly noone will admit to outright ignoring consumers in smart meter standardization their role is largely limited to the occassional public consultation. At the same time the standards are written by technologists–it seems largely without input on their practicality or socio-technological implications from fields such as HCI.

1.2.2 Common components

Smart meters usually are built around an off-the-shelf microcontroller. Some meters use specialized smart metering SOCs[53] while others use standard microcontrollers with core metering functions implemented in external circuitry (cf. sec. 3.3.1 where we detail the meter in our demonstration setup). Specialized SoCs usually contain a segment LCD driver along with some high-resolution analog-to-digital converters for the actual measurement functions. In many smart meter designs used outside of Germany the metering SoC will be connected to another full-featured SoC acting as the modem. At a casual glance this might seem to be a security measure, but it may be more likely that this is done to ease integration of one metering platform with several different communication stacks (e.g. proprietary sub-gigahertz wireless, powerline communication (PLC) or ethernet). In these architectures there is a clear line of functional demarcation between the metering SoC and the modem. As evidenced by over-the-air software update functionality (see e.g. Honeywell Smart Energy [77]) this does not however extend to an actual security boundary.

Energy usage is calculated by measuring both voltage and current at high resolution and then integrating the measurements. Current measurements are usually made with either a current transformer or a shunt in a four-wire configuration. Voltage is measured by dividing input AC down with a resistor chain. Both are integrated digitally using the MCU's time base as a reference.

Whereas legacy electromechanical energy meters only provided a display of aggregate energy use through a decimal counter as well as an indirect indication of power through a rotating wheel one of the selling points of smart meters is their ability to calculate advanced statistics on energy use. These statistics are supposed to help customers better target energy conservation measures though evidence of this happening is scarce.

In addition to the pure measurement and data aggregation functions in many deployments smart meters perform two additional functions. One is to serve as a gateway between the utility company's control systems and large controllable loads in the consumer's household for Demand-Side Management (DSM). In DSM the utility company can control when exactly a high-power device such as a water storage heater is turned on. To the customer the precise timing does not matter since the storage heater is set so that it has enough hot water in its reservoir at all times. The utility company however can use this degree of control to reduce load variations during temporary imbalances such as peaks. The efficiency gains realized with this system

translate into lower electricity prices for DSM-enabled loads for the customer. Traditionally DSM was realized on a local level using ripple control systems. In ripple control control data is coded by modulating a carrier at a low frequency such as 400 Hz on top of the regular mains voltage. These systems require high-power transmitters at tens of kilowatts and still can only bridge regional distances[58].

Another important additional function is that in some countries some smart meters can be used to remotely disconnect consumer households with outstanding bills. Using euphemisms such as *utility revenue protection*[81] or *reducing nontechnical losses*[14] while cynically claiming *Consumer Empowerment*[81] these systems allow an utility company to remotely disconnect a customer at any time. Whereas before smart metering this required either additional hardware or an expensive site visit by a qualified technician smart meters have ushered in an era of frictionless control[8].

1.2.3 Cryptographic coprocessors

Just like in legacy electricity meters in smart meters physical security is still a key component of the overall system design. Since in both types of meter cost depends on physical quantities being measured at the customer premises customers can save cost in case they are able to falsify the meter's measurements without being detected. For this reason both types of meters employ countermeasures against physical intrusion. Compared to high-risk devices such as card payment processing terminals or ATMs the tamper proofing used in smart meters is only basic. Common measures include sealing the case by irreversibly ultrasonically welding front and back plastic shells together or the use of security seals on the lid covering the input/output screw terminals. Low-tech attacks using magnets to saturate the current transformer's ferrite cores are detected using hall sensors[79, 76, 59].

German smart metering standards are unique in that they specify the use of a smartcard-like security module to provide transport encryption and other cryptographic services[29, 27].

1.2.4 Physical structure and installation

Smart meters are installed like traditional electricity meters. In Japan this means they are usually installed on an exterior wall and need to be resistant against weather and extreme environmental conditions (direct sunlight, high temperature, high humidity). In Germany the meter is always installed either indoors or in an outdoor utility closet that is sealed to keep out the weather. In most countries the meter is connected through large integrated screw terminals. In the US meters compliant with the domestic ANSI C12 standard are round and plug into a large socket that is wired into the house or apartment's electrical connection.

Modern smart meters are usually made with plastic cases. Ferraris meters often used cases stamped from sheet metal with glass windows on them. Smart meters now look much more like other modern electronic devices. A common construction style is to separate the case in a front and back half with both halves clipped or ultrasonically welded together. Ultrasonic welding gives a robust, airtight interface. This interface cannot easily be separated and re-connected

[8]Note that in some countries such as the UK non-networked mechanical prepayment meters did exist. In such systems the user inserts coins into a coin slot that activates a load switch at the household's main electricity connection. These systems were non-networked and did not allow for remote control. A disadvantage of such systems compared to modern *smart* systems are the high cost of the coin acceptor and the overhead of site visits required to empty the coin box.

without leaving visible traces, which helps with tamper evidence properties. As an industry-standard process common in various consumer goods ultrasonic welding is a cheap and accessible technology[59, 53].

Communication interfaces sometimes are brought out through regular electromechanical connectors but often also are optical interfaces. A popular style here is to use a regular UART connected to an LED/phototransistor optocoupler mounted on the side of the case. The user interface is usually limited to an LCD display. For cost and ingress protection smart meters rarely use mechanical buttons. Some smart meters use a phototransistor mounted behind the faceplate that can be activated with a flashlight as a crude contact-less input device[59].

All meters provide several options for security seals to be installed to detect opening of the meter or access to its terminal block. The shape and type of these security seals varies. Factory-installed seals are used to detect tampering of the meter itself while seals made by the utility during meter installation are used to guard the meter's terminal block and detect attempts at by-passing.

1.3 Regulatory frameworks around the world

Smart metering regulation varies from country to country as it is tightly coupled to the overall regulation of the electrical grid. The standardization of the physical form factor and metrological parameters of a meter is usually separate from the standardization of its *smart* functionality. Most countries base the standard for their meters' outwards-facing communication interface on a family of standards unified under the IEC as DLMS/COSEM. Employing this base protocol ountry-specific standardization only covers which precise variant of it is spoken and what features are supported.

1.3.1 International standards

The family of standards one encounters most in smart metering applications are IEC 62056 specifying the Device Language Message Specification (DLMS) and the Companion Specification for Electronic Metering (COSEM). DLMS/COSEM are application-layer standards describing a request/response schema similar to e.g. HTTP. DLMS/COSEM are mapped onto a multitude of wire protocols. They can be spoken over TCP/IP or mapped onto low-speed UART serial interfaces [113, 125]. Besides DLMS/COSEM there are a multitude of standards usually specifying how DLMS/COSEM are to be applied.

DLMS/COSEM show some amount of feature creep. They do not adhere to the age-old systems design adage that a tool should *do one thing and do it well*. Instead they try to capture the convex hull of all possible applications. This led to a complicated design that requires extensive additional specification and testing to maintain even basic interoperability. In particular in the area of transport security it becomes evident that the IEC as an electrical engineering standards body stretched their area of expertise and resorting to established standard protocols would have improved the situation[135]. Compared to industry-standard transport security the IEC standards provide a simplistic key management framework based on a static shared key with unlimited lifetime and provide sub-optimal transport security properties (e.g. lack of forward-secrecy).

1.3.2 The regulatory situation in selected countries

In this section we will give an overview of the situation in a number of countries. This list of countries is not representative and notably does not include any developing countries and is geographically biased. We selected these countries for illustration only and based our selection in a large part on the availability of information in a language we read. We will conclude this section with a summarization of common themes.

Germany

Germany standardized smart metering on a national level. Apart from the calibration standards applying to any type of meter smart meters are covered by a set of communications and security standards developed by the German Federal Office for Information Security (BSI). Germany mandates smart meter installations for newly constructed buildings and during major renovations but does not require most legacy residential installations to be upgraded. This is a consequence of a 2013 cost-benefit analysis that found these upgrades to be uneconomical for the majority of residential customers[38, 36, 75, 14].

The German standards strictly separate between metering and communication functions. Both are split into separate devices, the *meter* and the *gateway* (called emphsmart meter gateway in full and often abbreviated emphSMGW). One or several meters connect to a gateway through a COSEM-derived protocol. The communication interface between meter and gateway can optionally be physically unidirectional. An unidirectional interface eliminates any possibility of meter firmware compromise. The gateway contains a cryptographic security module similar to a smartcard[96] that is entrusted with signing of measurements and maintaining an authenticated and encrypted communication channel with its authorities. Security of the system is certified according to a Common Criteria process.

The German specification does not include any support for load switches outside of demand-side management as they are common in some other countries. It does not prohibit the installation of one behind the smart meter installation. This makes it theoretically possible for a utility company to still install a load switch to disconnect a customer, but this would be a spearate installation from the smart meter. In Germany there are significant barriers that have to be met before a utility company may cut power to a household[124]. The elision of a load switch means attacks on German meters will be limited in influence to billing irregularities and attacks using DSM equipment.

The Netherlands

The Netherlands were early to take initiative to roll out smart metering after its recognition by the European Commission in 2006[47, 41]. After overcoming political issuses the Netherlands were above the European median in 2018 having replaced almost half of all meters[47, 118]. Dutch smart meters are standardized by a consortium of distribution system operators. They integrate gateway and metrology functions into one device. The utility-facing interface is a IEC DLMS/COSEM-based interface over cellular radio such as GPRS or LTE[7]. Like e.g. the German standard, the Dutch standard precisely specifies all communication interfaces of the meter[57]. Another parallel is that the Dutch standard also does not cover any functionality for remotely disconnecting a household. This absence of a load switch limits attacks on Dutch smart meters to causing billing irregularities.

The UK

The UK is currently undergoing a smart metering rollout. Meters in the UK are nationally standardized to provide both Zigbee ZSE-based and IEC DLMS/COSEM connectivity. UK smart metering specifications are shared between electrical and gas meters. Different to other countries' specifications the UK national specifications require electrical meters to have an integrated load switch and gas meters to have an integrated valve. In the UK a significant number of consumers are subject to prepaid electricity contracts. Prepayment and credit functionality are also specified in the national smart metering standard, as is remote firmware update functionality. Outside communications in these standards is performed through a gateway (there called *communications hub*) that can be shared between several meters [120, 121, 119, 14, 113]. The combination of both gas and electricity metering into one family of standards and the exceptionally large set of *required* features make the UK regulations the maximalist among the ones in this section. The mandatory inclusion of both load switches and remote connectivity up to remote firmware update make it an interesting attack target.

Italy

Italy was among the first countries to legally mandate the widespread installation of smart meters in households. Italy in 2006 and 2007 by law set a starting date for the rollout in 2008[14]. The Italian electricity market was recently privatized. While the wholesale market and transmission network privatization has advanced the vast majority of retail customers continued to use the incumbent distribution system operator ENEL as their supplier[118]. This dominant position allowed ENEL to orchestrate the large-scale rollout of smart meters in Italy. Almost every meter in Italy had been replaced by a smart meter by 2018[118]. An unique feature of the Italian smart metering infrastructure is that it relies on Powerline Communication (PLC) to bridge distances between meters and cellular radio gateways[74].

Japan

Japan is currently rolling out smart metering infrastructure. Compared to other countries in Japan significant standardization effort has been spent on smart home integration.[3, 113, 14]. Japan has domestic standards (JIS) for metrology and physical dimensions. The TEPCO deployment currently being rolled out is based on the IEC DLMS/COSEM standards suite for remote meter reading in conjuction with the Japanese ECHONET protocol for the home-area network. Smart meters are connected to TEPCO's backend systems through the customer's internet connection, sub-gigahertz radio based on 802.15.4 framing, regular landline internet or PLC[78, 113].

A unique point in the Japanese utility metering landscape is that the current practice is monthly manual readings. In Japan residential utility meters are usually mounted outside the building on an exterior wall and every month someone with a mirror on a long stick will come and read the meter. The meter reader then makes a thermal paper print-out of the updated utility bill and puts it into the resident's post box. This practice gives consumers good control over their consumption but does incur significant pesonnel overhead.

The USA

In the USA the rollout of smart meters has been promoted by law as early as 2005. The US electricity market is highly complex with states having significant authority to decide on their

own policies[14]. Different from the IEC standards used in large fraction of the rest of the world, the USA have their own domestic set of standards for smart meters developed by ANSI[113]. The main difference between IEC and ANSI-standard meters is that ANSI-standard meters are round devices that plug into a wall-mounted socket while IEC devices are usually rectangular and connected directly to the mains wiring through large screw terminals[53].

1.3.3 Common themes

Researching the current situation around the world for the above sections we were able to distill some common themes. First, smart metering is slowly advancing on a global scale and despite significant reservations from privacy-conscious people and consumer advocates it seems it is here to stay. There are some notable exceptions of countries that have decided to scale-back an ongoing rollout effort after subsequent analysis showed economical or other issues[9].

The introduction of smart metering

The smart meter rollout is largely driven by utility companies. Utility companies field a variety of arguments for the rollout. The most prominent argument is a general increase in energy-efficiency along with a reduction of emissions. This argument is based on the estimation that smart metering will increase private customers' awareness of their own consumption and this will lead them to reduce their consumption. The second highly popular argument for smart metering is that it is necessary for the widespread adoption of renewable energies. This argument again builds on the trend towards *green* energy to rationalize smart metering. Often it is formulated as an *inevitability* instead of a choice.

Academic reception of smart metering is dyed with an almost unanimous enthusiasm. In particular smart meter communication infrastructure has received a large amount of research attention[58, 74, 80, 93, 97, 138, 5]. Outside of human-computer interaction claims that smart meters will reduce customer energy consumption have often been uncritically accepted.

Standardization and reality of smart devices

Regulators, utilities and academics meet in their enthusiasm on the issue of smart home integration of smart metering. A feature of many setups is that the meter acts as the centerpiece of a modern, fully integrated smart home[7, 70, 26, 1]. The smart meter serves as a communication hub between a new class of grid-aware loads and the utility company's control center. Large (usually thermal) loads such as dishwashers, refrigerators and air conditioners are forecasted to intelligently adapt their heating/cooling cycles to better match the grid's supply. A frequent scenario is that in which the meter bills the customer using near-real time pricing, and supplies large loads in the customer's household with this pricing information. These loads then intelligently schedule their operation to minimize cost[113]. At the time in the mid-2000nds when smart metering proposals were first advanced this vision might have been an effect of the *law of the instrument*[82]. Back then outside of specialty applications household devices were not usually networked[100]. Smart meters at the time may have seemed the obvious choice for a smart home communications hub.

From today's perspective, this idea is obviously outdated. Smart *things* now have found their way into many homes. Only these things are directly interconnected through the internet– foregoing the home-area network (HAN) technologies anticipated by the smart metering pioneers.

[9] cf. the Netherlands and Germany

The simple reason for this is that nowadays anyone has Wifi, and Wifi transceivers have become inexpensive enough to disappear in the bill of materials (BOM) cost of a large home device such as a washing machine. Smart meters are usually situated in the basement–physically far away from most of one's devices. This makes connecting them to said devices awkward and connecting them via the local Wifi lends the question why the smart devices should not simply use the internet in the first place.

Connecting things to a smart meter through a local bus is academically appealing. It promises cost-savings from a simpler physical layer (such as ZigBee instead of Wifi) and it neatly separates concerns into *home infrastructure* and the regular internet. Communication between smart meter and devices never leaves the house. This gives potential additional tolerance to utility backend systems breaking. It also physically keeps communication inside the house, bypassing the utility's eyes improving both customer privacy and agency. The presently popular model of a device as simple as a light switch proxying its every action through a manufacturer's servers somewhere on the public internet is in stark contrast to this scenario. Alas, the reason that this model is as popular is that in most cases it simply works. Device manufacturers simply integrate one of many off-the-shelf Wifi modules. The resulting device will work anywhere on earth[10]. A HAN-connected device would have several variants with different modems for different standards. Some might work across countries, but some might not. And in some countriese there might not even be a standard for smart grid HANs.

Looking at the situation like this begs the question why this realization has not yet found its way into mainstream acceptance by smart metering implementors. The customer-facing functionality promised through smart meters would be simple to implement as part of a now-standard *internet of things* application. An in-home display that shows real-time energy consumption and cost statistics would simply be an android tablet fetching summarized data from the utility's billing backend. Demand-side response by large loads would be as simple as an HTTP request with a token identifying the customer's contract that returns the electricity price the meter is currently charging along with a recommendation to switch on or off. It seems the smart home has already arrived while smart metering standardization is still getting off the starting blocks.

1.4 Security in smart distribution grids

The smart grid in practice is nothing more or less than an aggregation of embedded control and measurement devices that are part of a large control system. This implies that all the same security concerns that apply to embedded systems in general also apply to most components of a smart grid in some way. Where programmers have been struggling for decades now with input validation[92], the same potential issue raises security concerns in smart grid scenarios as well[101, 90]. Only, in smart grid we have two complicating factors present: Many components are embedded systems, and as such inherently hard to update. Also, the smart grid and its control algorithms act as a large (partially-)distributed system, making problems such as input validation or authentication difficult to implement[10] and adding a host of distributed systems problems on top[89].

Given that the electrical grid is a major piece of essential infrastructure in modern civilization, these problems amount to significant issues in practice. Attacks on the electrical grid may have grave consequences[5, 90] all the while the long maintenance cycles of various components make

[10]For some places channel assignments may have to be updated. This is a configuration-level change and in some devices is done by the end-user during provisioning.

the system slow to adapt. Thus, components for the smart grid need to be built to a much higher standard of security than most consumer devices to ensure they live up to well-funded attackers even decades down the road. This requirement intensifies the challenges of embedded security and distributed systems security among others that are inherent in any modern complex technological system. The safety-critical nature of modern smart metering ecosystems in particular was quickly recognized by security experts[5].

A point we will not consider in much depth is theft of electricity. An incentive for the introduction of smart metering that is frequently cited in utility industry publications outside of a general public's view is the reduction of electricity theft. Academic papers tend to either focus on other benefits such as generation efficiency gains through better forecasting or try to rationalize the funamentally anti-consumer nature of smart metering with strenuous claims of "enormous social benefits"[107]. Academics rarely point out the large economical incentive such *revenue protection* mechanisms provide[5].

This thesis will entirely focus on grid stability and discard electricity theft. For the attack scenarios we lay out billing inaccuracies of utility companies are of very low urgency compared to grid stability. In fact stability is a precondition for billing to happen. Additionally utility companies can already limit the volume of theft by cross-refrencing meter readings against trusted readings from upstream sections of the grid. This capability works even without smart meters and only gains speed from smart meters. A smart meter cannot prevent the customer from bypassing it with a section of wire. Due to the limit on its volume, electricity theft using smart meter hacking would not scale. Hackers would quickly be triangulated with no damage to consumers and limited damage to utility companies.

1.4.1 Privacy in the smart grid

A serious issue in smart metering setups is customer privacy. Even though the meter "only" collects aggregate energy consumption of a whole household this data is highly sensitive[102]. This counterintuitive fact was initially overlooked in smart meter deployments leading to outrage, delays and reduced features[47]. The root cause for this is that given sufficient timing resolution these aggregate measurements contain ample entropy. Through disaggregation individual loads can be identified and through pattern matching even complex usage patterns can be discerned with alarming accuracy[73]. Similar privacy issues arise in many other areas of modern life through pervasive tracking and surveillance[141]. What makes the case of smart metering worse is that even the fig leaf of consent such practices hide behind does not apply here. If I as a citizen do not consent to Google's privacy policy Google says I can choose not to use their service. In today's world this may not be a free choice making this argument totally invalid, but it is at least technically possible. Smart metering on the other hand is mandated by law. In some countries such as Germany a customer unwilling to accept the accompanying privacy violation cannot legally evade it[37].

1.4.2 Smart grid components as embedded devices

A fundamental challenge in smart grid implementations is the central role smart electricity meters play. Smart meters are used both for highly-granular load measurement and (in some countries) load switching[139]. Smart electricity meters are effectively consumer devices. They are built down to a certain price point that is measured by the burden it puts on consumers. The cost of a smart meter is ultimately limited by it being a major factor in the economies of a

smart meter rollout[38]. Cost requirements preclude some hardware features such as the use of a standard hardened software environment on a high-powerded embedded system (such as a hypervirtualized embedded linux setup) that would both increase resilience against attacks and simplify updates. Combined with the small market sizes in smart grid deployments [11] this produces a high cost pressure on the software development process for smart electricity meters.

1.4.3 The state of the art in embedded security

Embedded security generally is much harder than security of higher-level systems. This is due to a combination of the unique constraints of embedded devices (hard to update, usually small quantity) and their lack of capabilities (processing power, memory protection functions, user interface devices). Even very well-funded companies continue to have serious problems securing their embedded systems. A spectacular example of this difficulty is the recently-exposed flaw in Apple's iPhone SoC first-stage ROM bootloader[12], that allows a full compromise of any iPhone before the iPhone X. iPhone 8, one of the affected models, is still being manufactured and sold by Apple today[13]. In another instance, Samsung put a flaw in their secure-world firmware used for protection of sensitive credentials in their mobile phone SoCs in If both of these very large companies have trouble securing parts of their secure embedded software stacks measuring a mere few hundred bytes in Apple's case or a few kilobytes in Samsung's, what is a smart electricity meter manufacturer to do? For their mass-market phones, these two companies have R&D budgets that dwarf some countries' national budgets.

Since thorough formal verification of code is not yet within reach for either large-scale software development or code heavy in side-effects such as embedded firmware or industrial control software[106] the two most effective measures for embedded security is reducing the amount of code on one hand, and labour-intensively checking and double-checking this code on the other hand. A smart electricity manufacturer does not have a say in the former since it is bound by the official regulations it has to comply with, and will almost certainly not have sufficient resources for the latter.

[11] Most vendors of smart electricity meters only serve a handful of markets. For the most part, smart meter development cost lies in the meter's software There exist multiple competing standards applicable to various parts of a smart electricity meter. In addition, most countries have their own certification regimen[127]. This complexity creates a large development burden for new market entrants[131].

[12] Modern system-on-chips integrate one or several CPUs with a multitude of peripherals, from memory and DMA controllers over 3D graphics accelerators down to general-purpose IO modules for controlling things like indicator LEDs. Most SoCs boot from one of several boot devices such as flash memory, ethernet or USB according to a configuration set e.g. by connecting some SoC pins a certain way or set by device-internal write-only fuse bits.

Physically, one of the processing cores of the SoC (usually one of the main CPU cores) is connected such that it is taken out of reset before all other devices, and is tasked with switching on and configuring all other devices of the SoC. In order to run later intialization code or more advanced bootloaders, this core on startup runs a very small piece of code hard-burned into the SoC in the factory. This ROM loader initializes the most basic peripherals such as internal SRAM memory and selects a boot device for the next bootloader stage.

Apple's ROM loader performs some authorization checks, to ensure no unauthorized software is loaded. The present flaw allows an attacker to circumvent these checks, booting code not authorized by Apple on a USB-connected iPhone, compromising Apple's chain of trust from ROM loader to userland right at its root.

[13] i.e. at the time this paragraph was written, on

1.4.4 Attack avenues in the smart grid

If we model the smart grid as a control system responding to changes in inputs by regulating outputs, on a very high level we can see two general categories of attacks: Attacks that directly change the state of the outputs, and attacks that try to influence the outputs indirectly by changing the system's view of its inputs. The former would be an attack such as one that shuts down a power plant to decrease generation capacity. The latter would be an attack such as one that forges grid frequency measurements where they enter a power plant's control systems to provoke increasing oscillation in the amount of power generated by the plant according to the control systems' directions.

Communication channel attacks

Communication channel attacks are attacks on the communication links between smart grid components. This could be attacks on IP-connected parts of the core network or attacks on shared busses between smart meters and IP gateways in substations. Generally, these attacks can be mitigated by securing the aforementioned communication links using modern cryptography. IP links can be protected using TLS, and more low-level busses can be protected using more lightweight Noise[108]-based protocols. Cryptographic security transforms an attackers ability to manipulate communication contents into a mere denial of service attack. Thus, in addition to cryptographic security safety under DoS conditions must be ensured to ensure continued system performance under attacks. This safety property is identical with the safety required to withstand random outages of components, such as communications link outages due to physical damage from storms, flooding etc. In general, attacks at the meter level may be hard to weaponize since meters are used mostly for billing and forecasting purposes and for more critical grid control purposes there exist several additional layers of sensors above smart meters that limit how much an attacker can falsify smart meter readings without the manipulation being obvious. In order for an attack to have more far-reaching consequences the attacker would need to compromise additional grid infrastructure[83, 85].

Exploiting centralized control systems

The type of smart grid attack most often cited in popular discourse, and to the author's knowledge the only type that has so far been conducted in practice, is a direct attack on centralized control systems. In this attack, computer components of control systems are compromised by the same techniques used to compromise any other kind of computer system such as exploiting insecure services running on internet-exposed ports and using one compromised system to compromised other systems connected with it through an ostensably secure internal network. These attacks are very powerful as they yield the attacker direct control over whatever outputs the control systems are controlling. If an attacker manages to compromise a power stations control computers, they may be able to influence generation output or even cause an emergency shutdown.

Despite their potentially large impact, these attacks are only moderately interesting from a scientific perspective. For one, their mitigation mostly consists of a straightforward application of security practices well-known for decades. Though there is room for the implementation of genuinely new, application-specific security systems in this field, the general state of the art is lacking behind the rest of the computer industry such that the low-hanging fruit should take priority.

In addition, given political will these systems can readily be secured since there is only a

comparatively small number of them and driving a technician to every one of them in turn to install some security update is perfectly feasible.

Control function exploits

Control function exploits are attacks on the mathematical control loops used by the centralized control system. One example of such an attack would be resonance attacks as described in Wu et al. [137]. In this kind of attack, inputs from peripheral sensors indicating grid load to the centralized control system are carefully modified to cause a disproportionally large oscillation in control system action. This type of attack relies on complex resonance effects that arise when mechanical generators are electrically coupled. These resonances, coloquially called "modes" are well-studied in power system engineering[111, 72, 63]. Even disregarding modern attack scenarios, for stability electrical grids are designed with measures in place to dampen any resonances inherent to grid structure. Still, requiring an accurate grid model these resonances are hard to analyze and unlikely to be noiticed under normal operating conditions.

Mitigation of these attacks is most easily done by on the one hand ensuring unmodified sensor inputs to the control systems in the first place, and on the other hand carefully designing control systems not to exhibit exploitable behavior such as oscillations.

Endpoint exploits

One rather interesting attack on smart grid systems is one exploiting the grid's endpoint devices such as smart electricity meters[14] These meters are deployed on a massive scale, with several thousand meters deployed for every substation. Thus, once compromised restoration to an uncompromised state can be potentially very difficult if it requires physical access to thousands of devices hidden inaccessible in private homes.

By compromising smart electricity meters, an attacker can trivially forge the distributed energy measurements these devices perform. In a best-case scenario, this might only affect billing and lead to customers being under- or over-charged if the attack is not noticed in time. However, in a less ideal scenario the energy measurements taken by these devices migth be used to inform the grid centralized control systems and a falsification of these measurements might lead to inefficiency.

In some countries and for some customers, these smart meters have one additional function that is highly useful to an attacker: They contain high-current load switches to disconnect the entire household or business in case electricity bills are left unpaid for a certain period. In countries that use these kinds of systems, the load disconnect is often simply hooked up to one of the smart merter's central microcontroller's general-purpose IO pins, allowing anyone compromising this microcontroller's firmware to actuate the load switch at will.

Given control over a large number of network-connected smart meters, an attacker might thus be able to cause large-scale disruptions of power consumption by repeatedly disconnecting and re-connecting a large number of consumers. Combined with an attack method such as the resonance attack from Wu et al. [137] that was mentioned above, this scenario poses a serious danger to grid stability.

[14]Though potentially this could also aim at other kinds of devices distributed on a large scale such as sensors in unmanned substations.

1.4.5 Attacker models in the smart grid

1.4.6 Practical attacks

1.4.7 Practical threats

1.4.8 Conclusion, or why we are doomed

We can conclude that a compromise of a large number of smart electricity meters cannot be ruled out. The complexity of network-connected smart meter firmware makes it exceedingly unlikely that it is in fact flawless. Large-scale deployments of these devices under some circumstances such as where they are used with load disconnect relays make them an attractive target for attackers interested in causing grid instability. The attacker model for these devices very definitely includes enemy states, who have considerable resources at their disposal.

For a reasonable guarantee that no large-scale compromises of hard- and software built today will happen over a span of some decades, we would have to radically simplify its design and limit attack surface. Unfortunately, the complexity of smart electricity meter implementations mostly stems from the large list of requirements these devices have to conform with. Additionally, standards have already been written and changes that reduce scope or functionality have become exceedingly unlikely at this point.

A general observation with smart grid systems of any kind is that they comprise a zealous departure of the decentralized control structure of yesterday's dumb grid and the advent of centralization at an enormous scale. This modern, centralized infrastructure has been carefully designed to defend against malicious actorsand all involved parties have an interest in keeping it secure. Still, like in any other system this centralization also makes a very attractive target for attackers since an attacker can likewise employ this centralized control to their goals. Fundamentally, decentralized systems tend to make attacks of any kind a lot more costly and one might question whether security has truly been gained during smart grid rollout.

Chapter 2

Restoring endpoint safety in an age of smart devices

If as layed out in the previous paragraph we cannot rule out a large-scale compromise of smart energy meters, we have to rephrase our claim to security. If we cannot rule out exploitation, we have to limit its impact. If we assume that we cannot strip any functionality from smart meters since it may be required by standards or for enormous social benefits[107] all we can do is to flush out an attacker once they are in.

In a worst-case scenario an attacker would gain unconstrained code execution e.g. by exploiting a flaw in a network protocol implentation. Since smart meters use standard microcontrollers that do not have advanced memory protection functions (see pg. 1.2.2), at this point we can assume the attacker has full control over the main microcontroller. With this control they can actuate the load switch if present, transmit data through the device's communication interfaces or use the user interface components such as LEDs and the LCD. Using the self-programming capabilities of modern flash microcontrollers, an attacker may even gain persistency without much trouble. Note that in systems separating cryptographic functions into some form of cryptographic module such as systems used in Germany we can be optimistic and assume the attacker has not in fact compromised this cryptographic co-processor yet and does not have access to any cryptographic secrets yet.

Given that the attacker has complete control over the meter's core microcontroller and given that due to cost constraints we are bound to use whatever microcontroller the meter OEM has chosen for their design, we cannot rely on software running on the core mircocontroller to restore system integrity.

Our solution to this problem is to add another, very small microcontroller to the smart meter design. This microcontroller will contain a small piece of software to receive cryptographically authenticated commands from utility companies and on demand reset the meter's core microcontroller to a known-good state. We have to assume the code in the core controller's flash memory has been compromised, so our only option to flush out an attacker is to re-program the core microcontroller in its entirety. We propose using JTAG to re-program the core microcontroller with a known-good firmware image read from a sufficiently large SPI flash connected to the reset controller. JTAG is supported by most microcontrollers complex enough to end up in a smart meter design and given adequate documentation JTAG programming functionality can be ported to new microcontrollers with relatively little work.

On the microcontroller side our solution requires the JTAG interface to be activated (i.e. not fused-shut) and for our solution to work core microcontroller firmware must not be able

to permanently disable the JTAG interface from within. In microcontrollers that do not yet provide this functionality this is a minor change that could be added to a custom microcontroller variant at low cost. On most microcontrollers keeping JTAG open should not interfere with code readout protection. Code secrecy should be of no concern[115] here but besides security manufacturers have strong preferences about this due to fear of copyright infringement.

2.1 The theory of endpoint safety

In order to gain anything by adding our reset controller to the smart meter's already complex design we must satisfy two interrelated conditions.

1. SECURITY means our reset controller itself does not have any remotely exploitable flaws
2. SAFETY menas our reset controller will perform its job as intended

Note that our SECURITY property includes only remote exploitation, and excludes any form of hardware attack. Even though most smart meters provide some level of physical security, we do not wish to make any assumptions on this. In the following section we will elaborate our attacker model and it will become apparent that sufficient physical security to defend against all attackers in our model would be infeasible, and thus we will design our overall system to remain secure even assuming some number of physically compromised devices.

2.1.1 Attack characteristics

The attacker model these two conditions must hold under is as follows. We assume three angles of attack: Attacks by the customer themselves, attacks by an insider within the metering systems controlling utility company and lastly attacks from third parties. Examples for these third parties are hobbyist hackers or outside cyber-criminals on the one hand, but also other companies participating in the smart grid infrastructure besides the utility company such as intermediary providers of meter-reading services.

Due to the critical nature of the electrical grid, we have to include hostile state actors in our attacker model. When acting directly, these would be classified as third-party attackers by the above schema, but they can reasonably be expected to be able to assume either of the other two roles as well e.g. through infiltration or bribery. Fraunholz, Duque Anton, and Schotten [66] in their elaboration of their generalized attacker model give some classification of attackers and provide a nice taxonomy of attacker properties. In their threat/capability rating, criminals are still considered to have higher threat rating than state-sponsored attackers. The New York Times reported in 2016 that some states recruit their hacking personnel in part from cyber-criminals. If this report is true, in a worst-case scenario we have to assume a state-sponsored attacker to be the worst of both types. Comparing this against the other attacker types in Fraunholz, Duque Anton, and Schotten [66], this state-sponsored attacker is strictly worse than any other type in both variables. We are left with a highly-skilled, very well-funded, highly intentional and motivated attacker.

Based on the above classification of attack angles and our observations on state-sponsored attacks, we can adapt Fraunholz, Duque Anton, and Schotten [66] to our problem, yielding the following new attacker types:

1. **Utility company insiders controlled by a state actor** We can ignore the other internal threats described in Fraunholz, Duque Anton, and Schotten [66] since an insider cooperating with a state actor is strictly worse in every respect.

2. **State-sponsored external attackers** A state actor can directly attack the system through the internet.

3. **Customers controlled by a state actor** A state actor can very well compromise some customers for their purposes. They might either physically infiltrate the system posing as legitimate customers, or they might simply deceive or bribe existing customers into cooperation.

4. **Regular customers** Though a hostile state actor might gain control of some number of customers through means such as voluntary cooperation, bribery, infiltration, they are limited in attack scale since they do not want to arouse premature attention. Though regular customers may not have the motivation, skill or resources of a state-sponsored attacker, potentially large numbers of them may try to attack a system out of financial incentives. To allow for this possibility, we consider regular customers separate from state actors posing as customers in some way.

2.1.2 Overall structural system security

Considering overall security, we first introduce the *reset authority*, a trusted party acting as the single authority for issuing reset commands in our system. In practice this trusted party may be part of the utility company, part of an external regulatory body or a hybrid setup requiring both to cooperate. We assume this party will be designed to be secure against all of the above attacker types. The precise design of this trusted party is out of scope for this work but we will list some practical suggestions on how to achieve security below.

Using an asymmetric cryptographic design centered around the *reset authority*, we rule out all attacks except for denial-of-service attacks on our system by any of the four attacker types. All reset commands in our system originate from the *reset authority* and are cryptographically secured to provide authentication and tamper detection. Under this model, attacks on the electrical grid components between the *reset authority* and the customer device degrade into man-in-the-middle attacks. To ensure the SAFETY criterion from Section 2.1 holds we must make sure our cryptography is secure against man-in-the-middle attacks and we must try to harden the system against denial-of-service attacks by the attacker types listed above. Given our attacker model we cannot fully guard against this sort of attack but we can at least choose a commmunication channel that is resilient against denial of service attacks under the above model.

Finally, we have to consider the issue of hardware security. We will solve the problem of physical attacks on some small number of devices by simply not programming any secret information into these devices. This also simplifies hardware production. From consideration in this work we explicitly rule out any form of supply-chain attack as out-of-scope.

2.1.3 Complex microcontroller firmware

The SECURITY property from 2.1 is in a large part reliant on the security of our reset controller firmware. The best method to increase firmware security is to reduce attack surface by limiting

external interfaces as much as possible and by reducing code complexity as much as possible. If we avoid the complexity of most modern microcontroller firmware we gain another benefit beyond implicitly reduced attack surface: If the resulting design is small enough we may attempt formal verification of our security property. Though formal verification tools are not yet suitable for highly complex tasks they are already adequate for small amounts of code and simple interfaces.

2.1.4 Modern microcontroller hardware

Microcontrollers have gained enormously in both performance/efficiency as well as in peripheral support. Alas, these gains have largely been driven by insatiable customer demand for faster, more powerful chips and for a long time security has not been considered important outside of some specific niches such as smartcards. Traditionally a microcontroller would spend its entire lifetime without ever being exposed to any networks. Though this trend has been reversing with the increasing adoption of internet-of-things things and more advanced security features have started appearing in general-purpose microcontrollers, most still lack even basic functionality found in processors for computers or smartphones.

One of the components lacking from most microcontrollers is strong memory protection or even a memory mapping unit as it is found in all modern computer processors and SoCs for applications such as smartphones. Without an MPU/MPU some mitigations for memory safety violations cannot be implemented. This and the absence of virtualization tools such as ARM's TrustZone make hardening microcontroller firmware a big task. It is very important to ensure memory safety in microcontroller firmware through tools such as defensive coding, extensive testing and formal verification.

In our design we achieve simplicity on two levels: One, we isolate the very complex metering firmware from our reset controller by having both run on separate microcontrollers. Two, we keep the reset controller firmware itself extremely simple to reduce attack surface there.

2.1.5 Regulatory and economical constraints

2.1.6 Safety vs. Security: Opting for restoration instead of prevention

By implementing our reset system as a physically separate microcontroller we sidestep most security issues around the main application microcontroller. There are some simple measures that can be taken to harden this firmware. Implementing industry best practices such as memory protection or stack canaries will harden the system and increase the cost of an attack but it will not yield a system that we can be confident enough in to say it is fully secure. The complexity of the main application controller firmware makes fully securing the system a formidable effort–and one that would have to be repeated by every meter vendor for every one of their code bases.

In contrast to this our reset system does not provide any additional security. Any attack that could occur without it can still occur with it in place. What it provides is a fail-safe mechanism that can quickly immobilize a malicious actor even mid-attack. It does this in a way that can be adapted to any meter architecture and any microcontroller platform with low effort since it relies on established standard interfaces such as JTAG and SWD. Concentrating research and development resources on a single platform like this allows for a system that is more economical to implement across device series and across vendors.

Attack resilience in the power grid can benefit from a safety-focused approach. The greater danger such an attack poses is not the temporary denial of service of utility metering functions. Even in a highly integrated smart grid as envisioned by utility companies their measurement

functions are used by utility companies to increase efficiency and reduce cost but are not necessary for the grid to function at all. Thus if we can provide mere *safety* with a fail-safe semantic instead of unattainable perfect *security* we have gained resilience against a large class of realistic attack scenarios.

2.1.7 Technical outline of a safety reset system

There are several ways our system could be practically implemented. The most basic way is to add a separate microcontroller connected to the meter's main application MCU and optionally other embedded microcontrollers such as modems. This discrete chip could either be placed on the metering board itself or it could be placed on a separate PCB connected to the programming interface(s) of the metering board. In certain cases the latter might allow use in otherwise unmodified legacy designs.

The saftey reset controller would be a much simpler MCU than the meter's main application controller. Its software can be held simple leading to low program flash and RAM requirements. Since it does not need to address rich periphery such as external parallel memory, LCDs etc. it can be a physically small, low-pin count device. If the main application controller is supposed to be reset to a full factory image with little or no reduced functionality its firmware image size is certainly too large for the reset controller's embedded flash. Thus a realistic setup would likely use an external SPI flash chip to store this image.

The most likely interfaces to reset the main application controller and possibly other microcontrollers such as modem chips would be the controller's integrated programming port such as JTAG. There exist a variety of programming interfaces for microcontrollers but for moderately complex ones JTAG has grown to be by far the most broadly supported one. Parallel high-voltage flash programming has come to be uncommon in modern microcontrollers and most chips nowadays use some form of a serial interface. Some vendors have their own proprietary serial in-system programming interfaces that they use on certain parts instead of or in addition to JTAG. The reasons for this usually are either lower complexity in parts that do not require full debugging capabilities as provided by JTAG or the high pin count of JTAG.

The kind of microcontroller that would likely be used as the main application controller in a smart meter application will almost certainly support JTAG. These microcontrollers are high pin-count devices since they need to connect to a large set of peripherals such as the LCD and the large program flash makes it likely for a proper debugging interface to be present.

The one remaining issue in this coarse technical outline is what communication interface should be used to transmit the trigger command to the reset controller. In the following section we will give an overview on communication interfaces established in energy metering applications and evaluate each of them for our purpose.

2.2 Communication channels on the grid

There is a number of well-established technologies for communication on or along power lines. We can distinguish three basic system categories: Systems using separate wires (such as DSL over landline telephone wiring), wireless radio systems (such as LTE) and *powerline communication* (PLC) systems that re-use the existing mains wiring and superimpose data transmissions on the 50 Hz mains sine[74, 80].

For our scenario, we will ignore short-range communication systems. There exists a large number of *wideband* powerline communication systems that are popular with consumers for

bridging ethernet between parts of an apartment or house. These systems transmit at up to several hundred megabits over distances up to several tens of meters[80]. Technologically, these wideband PLC systems are very different from *narrowband* systems used by utilities for load management among other applications and they are not relevant to our analysis.

2.2.1 Powerline communication (PLC) systems and their use

In long-distance communications for applications such as load management, PLC systems are attractive since they allow re-using the existing wiring infrastructure and have been used as early as in the 1930s[67]. Narrowband PLC systems are a potentially low-cost solution to the problem of transmitting data at small bandwidth over distances of several hundred meters up to tens of kilometers.

Narrowband PLC systems transmit on the order of kilobits per second or slower. A common use of this sort of system are *ripple control* systems. These systems superimpose a low-frequency signal at some few hundred Hertz carrier frequency on top of the 50Hz mains sine. This low-frequency signal is used to encode switching commands for non-essential residential or industrial loads. Ripple control systems provide utilities with the ability to actively control demand while promising small savings in electricity cost to consumers[58].

In any PLC system there is a strict tradeoff between bandwidth, power and distance. Higher bandwidth requires higher power and reduces maximum transmission distance. Where ripple control systems usually use few transmitters to cover the entire grid of a regional distribution utility, higher-bandwidth bidirectional systems used for automatic meter reading (AMR) in places such as italy or france require repeaters within a few hundred meters of a transmitter.

2.2.2 Landline and wireless IP-based systems

Especially in automated meter reading (AMR) infrastructure the cost-benefit tradeoff of powerline systems does not always work out for utilities. A common alternative in these systems is to use the public internet for communication. Using the public internet has the advantage of low initial investment on the part of the utility company as well as quick commissioning. Disadvantages compared to a PLC system are potentially higher operational costs due to recurring fees to network providers as well as lower reliability. Being integrated into power grid infrastructure, a PLC system's failure modes are highly correlated with the overall grid. Put briefly, if the PLC interface is down, there is a good chance that power is out, too. In contrast to this general internet services exhibit a multitude of failures that are entirely decorrelated from power grid stability.

For purposes such as meter reading for billing purposes, this stability is sufficient. However for systems that need to hold up in crisis situations such as the recovery system we are contemplating in this thesis, the public internet may not provide sufficient reliability.

2.2.3 Short-range wireless systems

Smart meters contain copious amonuts of firmware but still pale in comparison to the complexity of full-scale computers such as smartphones. For short-range communication between a meter and a cellular radio gateway mounted nearby or between a meter an an meter reading operator in a vehicle on the street a protocol such as Wifi (802.11) might be too complex in most cases. Absent widely-used standards in this space proprietary radio protocols instead grow very

attractive. These might be based on some standardized lower-level protocol such as ZigBee (802.15) or might be entirely home-grown. To a meter manufacturer a proprietary radio protocol has several advantages. It is easy to implement and requires zero external certification. It can be customized to its specific application. In addition it provides some level of vendor lock-in to customers sharing infrastructure such as a cellular radio gateway between multiple devices. In other fields where a lack of standardization has led to a proliferation of proprietary protocols such as home automation this has led to a fragmented protocol landscape. In other fields this is a large problem since consumer cannot easily integrated products made by different manufacturers into one system. In advanced metering infrastructure this is unlikely to be a disadvantage since ususally there is only one distribution grid operator for an area. Additionally shared resources such as a cellular radio gateway would most likely only be shared within a single building and within a single building usually all meters are operated by the same provider.

Systems in Europe commonly support Wireless M-Bus, an european standardized protocol[134] that operates on several ISM bands[1]. ZigBee is another popular standard and some vendors additionally support their own proprietary protcols[2].

2.2.4 Frequency modulation as a communication channel

For our system, we chose grid frequency modulation (henceforth GFM) as a low-bandwidth uni-directional broadcast communications channel. Compared to traditional PLC GFM requires only a small amount of additional hardware, works reliably throughout the grid and is harder to manipulate by a malicious actor.

Grid frequency in europe's synchronous areas is nominally 50 Hertz, but there are small load-dependent variations from this nominal value. Any device connected to the power grid (or even just within physical proximity of power wiring) can reliably and accurately measure grid frequency at low hardware overhead. By intentionally modifying grid frequency, we can create a very low-bandwidth broadcast communication channel. Grid frequency modulation has only ever been proposed as a communications channel at very small scales in microgrids before[132] but to our knowledge has not yet been considered for large-scale application.

Advantages of using grid frequency for communication are low receiver hardware complexity as well as the fact that a single transmitter can cover an entire synchronous area. Though the transmitter has to be very large and powerful, setup of a single large transmitter faces lower bureaucratic hurdles than integration of hundreds of smaller ones into hundreds of local systems each with autonomous goverance.

The frequency dependency of grid frequency

Despite the awesome complexity of large power grids the physics underlying their response to changes in load and generation is surprisingly simple. Individual machines (loads and generators) can be approximated by a small number of differential equations and the entire grid can be modelled by aggregating these approximations into a large system of nonlinear differential equations. Evaluating these systems it has been found that in large power grids small-signal steady-state changes in generation/consumption power balance cause an approximately linear

[1]Frequency bands that can be used for *Industrial, Scientific and Medical* applications by anyone and that do not require obtaining a license for transmitter operation. Manufacturers can use whatever protocol they like on these bands as long as they obtain certification that their transmitters obey certain spectral and power limitations.

[2]For an example see Honeywell Smart Energy [77]

change in frequency[87, 46, 130, 129]. *Small signal* here describes changes in power balance that are small compared to overall grid power. *Steady state* describes changes over a timeframe of multiple cycles as opposed to transient events that only last a few milliseconds.

This approximately linear relationship allows the specification of a coefficient linking ΔP and Δf with unit $\mathrm{W\,Hz^{-1}}$. In this thesis we are using the European power grid as our model system. We are using data provided by ENTSO-E (formerly UCTE), the governing association of european transmission system operators. In our calculations we use data for the continental european synchronous area, the largest synchronous area. $\frac{\Delta P}{\Delta f}$, called *Overall Network Power Frequency Characteristic* by ENTSO-E is around $25\,\mathrm{GW\,Hz^{-1}}$.

We can derive general design parameter for any system utilizing grid frequency as a communications channel from the policies of ENTSO-E[130, 64]. Probably any such system should stay below a modulation amplitude of 100 mHz which is the threshold defined in the ENTSO-E incidents classification scale for a Scale 0-1 (from "Anomaly" to "Noteworthy Incident" scale) frequency degradation incident[64] in the continental europe synchronous area.

Control systems coupled to grid frequency

The ENTSO-E Operations Handbook Policy 1 chapter defines the activation threshold of primary control to be 20 mHz. Ideally a modulation system would stay well below this threshold to avoid fighting the primary control reserve. Modulation line rate should probably be on the order of a few hundred millibaud. Modulation at such high rates would outpace primary control action which is specified by ENTSO-E as acting within between “a few seconds” and 15 s.

The effective *Network Power Frequency Characteristic* of primary control in the european grid is reported by ENTSO-E at around $20\,\mathrm{GW\,Hz^{-1}}$. Keeping modulation amplitude below this threshold would help to avoid spuriously triggering these control functions. This works out to an upper bound on modulation power of $20\,\mathrm{MW\,mHz^{-1}}$.

An outline of practical transmitter implementation

In its most basic form a transmitter for grid frequency modulation would be a very large controllable load connected to the power grid at a suitable vantage point. A spool of wire submerged in a body of cooling water (such as a small lake with a fence around it) along with a thyristor rectifier bank would likely suffice to perform this function during occassional cybersecurity incidents. We can however decrease hardware and maintenance investment even further compared to this rather uncultivated solution by repurposing regular large industrial loads to our transmitter purposes in an emergency situation. For some preliminary exploration we went through a list of energy-intensive industries in Europe[60]. The most electricity-intensive industries in this list are primary aluminium and steel production. In primary production raw ore is converted into raw metal for further refinement such as casting, rolling or extrusion. In steelmaking iron is smolten in an electric arc furnace. In aluminium smelting aluminium is electrolytically extracted from alumina. Both processes involve large amounts of electricity with electricity making up 40 % of production costs. Given these circumstances a steel mill or aluminium smelter would be good candidates as transmitters in a grid frequency modulation system.

In aluminium smelting high-voltage mains is transformed, rectified and fed into about 100 series-connected cells forming a *potline*. Inside the pots alumina is dissolved in molten cryolite electrolyte at about 1000 °C and electrolysis is performed using a current of tens or hundreds of

kiloampere. Resulting pure aluminium settles at the bottom of the cell and is tapped off for further processing.

Like steelworks, aluminium smelters are operated night and day without interruption. Aside from metallurgical issues the large thermal mass and enormous heating power requirements do not permit power-cycling. Due to the high costs of production inefficiencies or interruptions the behavior of aluminium smelters under power outages is a fairly well-characterized phenomenon in the industry. The recent move away from nuclear power and to renewable energy has lead to an increase in fluctuations of electricity price throughout the day. These electricity price fluctuations have provided enough economic incentive to aluminium smelters to develop techniques to modulate smelter power consumption without affecting cell lifetime or the output product[55, 61]. Power outages of tens of minutes up to two hours reportedly do not cause problems in aluminium potlines and are in fact part of routine operation for purposes such as electrode changes[61, 105].

The power supply system of an aluminium plant is managed through a highly-integrated control system as keeping all cells of a potline under optimal operating conditions is challenging. Modern power supply systems employ large banks of diodes or SCRs to rectify low-voltage AC to DC to be fed into the potline[8]. The potline voltage can be controlled almost continuously through a combination of a tap changer and a transductor. The individual cell voltages can be controlled by changing the anode to cathode distance (ACD) by physically lowering or raising the anode. The potline power supply is connected to the high voltage input and to the potline through isolators and breakers.

In an aluminium smelter most of the power is sunk into resistive losses and the electrolysis process. As such an aluminium smelter does not have any significant electromechanical inertia compared to the large rotating machines used in other industries. Depending on the capabilities of the rectifier controls high slew rates should be possible, permitting modulation at high[3] data rates.

Avoiding dangerous modes

Modern power systems are complex electromechanical systems. Each component is controlled by several carefully tuned feedback loops to ensure voltage, load and frequency regulation. Multiple components are coupled through transmission lines that themselves exhibit complex dynamic behavior. The overall system is generally stable, but may exhbit some instabilities to particular small-signal stimuli[87, 46]. These instabilities, called *modes* occur when due to mis-tuning of parameters or physical constraints the overall system exhibits oscillation at particular frequencies. Kundur [87] split these into four categories:

Local modes where a single power station oscillates in some parameter

Interarea modes where subsections of the overall grid oscillate w.r.t. each other due to weak coupling between them

[3] Aluminium smelter rectifiers are *pulse rectifiers*. This means instead of simply rectifying the incoming three-phase voltage they use a special configuration of transformer secondaries and in some cases additional coils to produce a large number (such as 6) of equally spaced phases. Where a direct-connected three-phase rectifier would draw current in 6 pulses per cycle a pulse rectifier draws current in more, smaller pulses to increase power factor. E.g. a 12-pulse rectifier will draw current in 12 pulses per cycle. In the best case an SCR pulse rectifier switched at zero crossing should allow 0 % to 100 % load changes from one rectifier pulse to the next, i.e. within a fraction of a single cycle.

Control modes caused by imperfectly tuned control systems

Torsional modes that originate from electromechanical oscillations in the generator itself

The oscillation frequencies associated with each of these modes are usually between a few tens of Millihertz and a few Hertz, see for example Grebe et al. [72] and ENTSO-E System Protection Dynamics and WG [63]. It is hard to predict the particular modes of a power system at the scale of the central-european interconnected system. Theoretical analysis and simulation may give rough indications but cannot yield conclusive results. Due to the obvious danger as well as high economical impact due to inefficiencies experimental measurements are infeasible. Finally, modes are highly dependent on the power grid's structure and will change with changes in the power grid over time. For all of these reasons, a grid frequency modulation system must be designed very conservatively without relying on the absence (or presence) of modes at particular frequencies. A concrete design guideline that we can derive from this situation is that the frequency spectrum of any grid frequency modulation system should not exhibit any notable peaks and should avoid a concentration of spectral energy in certain frequency ranges.

Overall system parameters

In conclusion we end up with the following tunable parameters for a grid frequency modulation based on a large controllable load:

Modulation amplitude proportionally related to modulation power. In a practical setup we might realize a modulation power up to a few hundred MW which would yield maybe a few tens of mHz of frequency amplitude.

Modulation pre-emphasis and slew-rate control . Pre-emphasis might be necessary to ensure an adequate Signal-to-Noise ratio (SNR) at the receiver. Slew-rate control and other shaping measures might be necessary to reduce the impact of these sudden load changes on the transmitter's primary function (say, aluminium smelting) and to prevent disturbances to grid components.

Modulation frequency . For a practical implementation a careful study would be necessary to determine an optimal frequency band for operation. On one hand we need to prevent disturbances to the grid such as through excitation of some local or inter-area modes. On the other hand we need to optimize Signal-to-Noise ratio (SNR) and data rate to achieve optimal latency between transmission start and successful reception and to reduce the overall burden on transmitter and grid.

Further modulation parameters . The modulation itself has numerous parameters that are discussed in sec. 2.3.2 below.

2.3 From grid frequency to a reliable communication channel

2.3.1 Channel properties

In this section we will explore how we can construct a reliable communication channel from the analog primitive we outline in the previous section. Our load control approach to grid frequency modulation leads to a channel with the following properties.

Slow-changing. Accurate grid frequency measurements need several periods of the mains sine wave. Faster sampling rates can be achieved with more complex specialized synchrophasor estimation algorithms but this will result in a tradeoff between sampling rate and accuracy[9].

Analog. Grid frequency is an analog signal.

Noisy. While stable over long periods of time thanks to Load-Frequency Control[129] it shows considerable random short-term variations. In addition our modulation amplitude is limited by technical and economic constraints so we have to find a system that will work at poor SNRs.

Polarized. Grid frequency measurements have an inherent sense of *up* (higher frequencies). We can use this in a polarized modulation scheme to encode information without first transmitting some reference signal to establish this polarization.

2.3.2 Modulation and its parameters

In this section we will consider how to select a good set of parameters for a modulation scheme fitting grid frequency modulation.

The sensitivity of the grid to oscillation at particular frequencies described above means we should avoid any modulation technique that would concentrate a lot of energy in a small bandwidth. Taking this principle to its extreme provides us with a useful pointer towards techniques that might work well: Spread-spectrum techniques. By employing spread-spectrum modulation we can produce an almost ideal frequency-domain behavior that spreads the modulation energy almost flat across the modulation bandwidth[71] while at the same time achieving some modulation gain, increasing system sensitivity. This modulation gain spread-spectrum techniques yield potentially allows us to use a weaker stimulus, allowing further reduction of the probability of disturbance to the overall system. Spread-spectrum techniques also inherently allow us to tune the tradeoff between receiver sensitivity and data rate. This tunability is a highly useful parameter to have for the overall system design.

Spread spectrum covers a whole family of techniques. Goiser [71] separates these techniques into the coarse categories of *Direct Sequence Spread Spectrum*, *Frequency Hopping Spread Spectrum* and *Time Hopping Spread Spectrum*.

Goiser [71] assumes a BPSK or similar modulation underlying the spread-spectrum technique. Our grid frequency modulation channel effectively behaves more like a DC-coupled wire than a traditional radio channel: Any change in excitation will cause a proportional change in the receiver's measurement. Using our fft-based measurement methodology we get a real-valued signed quantity. In this way grid frequency modulation is similar to a channel using coherent modulation. We can transmit not only signal strength, but polarity too.

For our purposes we can discount both Time and Frequency Hopping Spread Spectrum techniques. Time hopping aids to reduce interference between multiple transmitters but does not help with SNR any more than Direct Sequence does. Our system is strictly limited to a single transmitter so we do not gain anything through Time Hopping.

Frequency Hopping Spread Spectrum techniques require a carrier. Grid frequency modulation itself is very limited in peak frequency deviation Δf. Frequency hopping could only be implemented as a second modulation on top of GFM, but this would not yield any benefits while increasing system complexity and decreasing data bandwidth.

Direct Sequence Spread Spectrum is the only remaining approach for our application. Direct Sequence Spread Spectrum works by directly modulating a long pseudorandom bit sequence onto the channel. The receiver must know the same pseudo-random bit sequence and continuously calculates the correlation between the received signal and the pseudo-random template sequence mapped from binary $[0, 1]$ to bipolar $[1, -1]$. The pseudorandom sequence has approximately equal number of 0 and 1 bits the correlation between the sequence and uncorrelated noise is small. The positive contribution of the $+1$ terms of the correlation template approximately cancel out with the -1 terms when multiplied with an uncorrelated signal such as white gaussian noise or another pseudo-random sequence.

By using a family of pseudo-random sequences with low cross-correlation channel capacity can be increased. Either the transmitter can encode data in the choice of sequence or multiple transmitters can use the same channel at once. The longer the pseudo-random sequence the lower its cross-correlation with noise or other pseudorandom sequences of the same length. Choosing a long sequence we increase modulation gain while decreasing bandwidth. For any given application the sweet spot will be the shortest sequence that is long enough to yield sufficient SNR for subsequent processing layers such as channel coding.

A popular code used in many DSSS systems are Gold codes. A set of Gold codes has small cross-correlations. For some value n a set of Gold codes contains $2^n + 1$ sequences of length $2^n - 1$. Gold codes are generated from two different maximum length sequences generated by linear feedback shift registers (LFSRs). For any bit count n there are certain empirically determined preferred pairs of LFSRs that produce Gold codes with especially good cross-correlation. The $2^n + 1$ gold codes are defined as the XOR sum of both LFSR sequences shifted from 0 to $2^n - 1$ bit as well as the two individual LFSR sequences. Given LFSR sequences `a` and `b` in numpy notation this is `[a, b] + [ a ^ np.roll(b, shift) for shift in len(b) ]`.

In DSSS modulation the individual bits of the DSSS sequence are called *chips*. Chip duration determines modulation bandwidth[71]. In our system we are directly modulating DSSS chips on mains frequency without an underlying modulation such as BPSK as it is commonly used in DSSS systems.

2.3.3 Error-correcting codes

To make our overall system reliable we have to layer some channel coding on top of our DSSS modulation. The messages we expect to transmit are at least a few tens of bits long. We are highly constrained in SNR due to limited transmission power. With lower SNR comes higher BER (bit error rate). Packet error rate grows exponentially with transmission length. For our relatively long transmissions we would realistically get unacceptable error rates.

Error correcting codes are a very broad field with many options for specialization. Since we are implementing nothing more than a prototype in this thesis we chose to not expend resources on optimization too much and settled for a comparatively simple low-density parity check code. The state of the art has advanced considerably since the discovery of general LDPC codes. The main areas of improvement are overhead and decoding speed. Since transmission length in our system limits system response time but we do not have a fixed target there we can tolerate some degree of sub-optimal overhead. Decoding speed is of no concern to us as our data rate is extremely low.

An important concern for our prototype implementation was the availability of reference implementations of our error correcting code. We need a python implementation for test signal generation on a regular computer and we need a small C or C++ implementation that we can

adapt to embedded firmware. LDPC codes are a popular textbook example of error-correcting codes and we had no particular difficulty finding either.

2.3.4 Cryptographic security

Informally the system we are looking for can be modelled as consisting of three parties: The trusted Transmitter, one of a large number of untrusted Receivers, and an Attacker. These three play according to the following rules:

1. Transmitter and Attacker can both transmit any bit sequence
2. Receiver receives any transmission by either Transmitter or Attacker but cannot distinguish between the two on the signal level
3. Attacker knows anything a Receiver might know
4. Transmitter is stronger than Attacker and will "win" in simultaneous transmission
5. Both Transmitter and Receiver can be seeded with some information on each other such as public key fingerprints.

We are not interested in congestion scenarios where an attacker attempts to disrupt an ongoing transmission by the transmitter. In practice there are several avenues to prevent such attempts including the following. Compromised loads that are being abused by the attacker can be manually disconnected by the utility. Error-correcting codes can be used to provide resiliency against small-scale disturbances. Finally, the transmitter can be designed to have high enough power to be able to override any likely attacker.

Our goal is to find a cryptographic primitive that has the following properties:

1. Transmitter can produce a transmission bit sequence $\mathbf{s}$ (or equivalently a set of such sequences) that Receiver can uniquely identify as being generated by Transmitter: $\mathcal{R}(\mathbf{s}) = 1$. Upon reception of this sequence, Receiver performs the safety reset.
2. Attacker cannot forge $\mathbf{s}$, that is find $\mathbf{s}'$ such that $\mathbf{s} \neq \mathbf{s}' \wedge \mathcal{R}(\mathbf{s}') = 1$
3. Our system conforms to an at-most-once semantic. That is, upon transmission of a valid bit sequence coded for a particular Receiver or set of receivers each one either performs exactly one safety reset or none at all. We cannot achieve an exactly-once semantic since we are using an unidirectional lossy communication primitive. More coloquially, Receiver might be offline due to a localized power outage and might thus not hear Transmitter even if our broadcast primitive is reliable. The practical impact of this limitation can be mitigated by transmitter simply repeating itself until the desired effect has been achieved.

An important limitation from the rules of our setup above is that Attacker can always record the bit sequence Transmitter transmits and replay that same sequence later. Before considering any cryptographic approaches we can make the preliminary observation that we can trivially prevent Attacker from violating the at-most-once criterion by simply requiring Receiver to memorize all bit sequences that have been transmitted thus far and only reacting to new bit sequences. This means an attacker might be able to cause offline receivers to reset at

a later point, but considering our goal is to reset them in the first place this would not pose a danger to the system.

As it seems we need a cryptographic primitive that looks somewhat like a signature. Different from a signature however, we have somewhat relaxed constraints here: While cryptographic signatures need to work over arbitrary inputs, all we want to "sign" here is the instruction to perform a safety reset. Since this is the only message we might ever want to transmit, our message space has only one entry and thus the informational content of our message is 0 bit! All the information we want to transmit is already encoded *in the fact that we are transmitting*, and we do not require any further payload to be transmitted. This means we can omit the entirety of the message and just transmit whatever "signature" we produce. This is useful since we have to conserve transmission bits so our transmissions do not take exceeedingly long time over our extremely slow communication channel.

We could use any of several traditional asymmetric cryptographic primitives to produce these signatures. The comparatively high computational effort required for signature verification would not be an issue. Transmissions take several minutes anyway and we can afford to spend some tens of seconds even in signature verification. Transmission length and by proxy system latency would be determined by the length of the signature. For RSA signature length is the modulus length (i.e. larger than 1000 bit for even basic contemporary security). For elliptic curve-based systems signature size is approximately twice the curve length (i.e. 300 bit for contemporary security). However, we can do better than this: We can exploit the strange nature of our setting that our effective message entropy is 0 bit to derive a more efficient scheme.

Lamport signatures

In 1979, Lamport [88] introduced a signature scheme that is based only on a one-way function such as a cryptographic hash function. The basic observation is that by choosing a random secret input to a one-way function and publishing the output, one can later prove knowledge of the input by simply publishing it. In the following paragraphs we will describe a construction of a one-time signature scheme based on this observation. The scheme we describe is the one usually called a "Lamport Signature" in modern literature and is slightly different from the variant described in the 1979 paper, but for our purposes we can consider both to be equivalent.

Setup. In a Lamport signature, for an n-bit hash function H the signer generates a private key $s = \left(s_{b,i} | b \in \{0,1\}, 0 \leq i < n\right)$ of $2n$ random strings of length n. The signer publishes a public key $p = \left(p_{b,i} = H\left(s_{b,i}\right), b \in \{0,1\}, 0 \leq i < n\right)$ that is simply the list of hashes of each of the random strings that make up the private key.

Signing. To sign a message m, the signer publishes the signature $\sigma = \left(\sigma_i = k_{H(m)_i,i}\right)$ where $H(m)_i$ is the i-th bit of H applied to m. That is, for the i-th bit of the message's hash $H(m)$ the signer publishes either of $p_{0,i}$ or $p_{1,i}$ depending on the hash bit's value, keeping the other entry of P secret.

Verification. The verifier can compute $H(m)$ themselves and check the corresponding entries $\sigma_i = k_{H(m)_i}$ of S correctly evaluate to $p_{b,i} = H\left(s_{b,i}\right)$ from P under H.

The above scheme is a one-time signature scheme only. After one signature has been published for a given key, the corresponding key must not be re-used for other signatures. This is intutively clear as we are effectively publishing part of the private key as the signature, and if we were

to publish a signature for another message an attacker could derive additional signatures by "mixing" the two published signatures.

Winternitz Signatures

An improvement to basic Lamport signatures as described above are Winternitz signatures as detailed in Merkle [99] and Dods, Smart, and Stam [54]. Winternitz signatures reduce public key length as well as signature length for hash length n from $2n$ to $\mathcal{O}\left(n/t\right)$ for some choice of parameter t (usually a small number such as 4).

Setup. The signer generates a private key $s = (s_i)$ consisting of $\lceil \frac{n}{t} \rceil$ random bit strings. The signer publishes a public key $p = \left(H^{2^t}\left(s_i\right)\right)$ where each element $H^{2^t}\left(s_i\right)$ is the 2^t-fold recursive application of H to s_i.

Signing. The signer splits m padded to a multiple of t bits into $\lceil \frac{n}{t} \rceil$ chunks m_i of t bit each. The signer publishes the signature $\sigma = \left(\sigma_i = H^{m_i}\left(s_i\right)\right)$.

Verification. The verifier can calculate for each $\sigma_i = H^{m_i}\left(s_i\right)$ that $H^{2^t - m_i}\left(\sigma_i\right) = H^{2^t - m_i}\left(H^{m_i}\left(s_i\right)\right) = H^{2^t - m_i + m_i}\left(s_i\right) = p_i$.

To prevent an attacker from forging additional signatures from one signature by calculating $\sigma'_i = H\left(\sigma_i\right)$ matching $m'_i = m_i + 1$, this scheme is usually paired with a simple checksum as described in Merkle [99].

Using hash-based signatures for trigger authentication

The most basic possible trigger authentication scheme would be to simply generate a random bit string secret key s and publish $p = H(s)$ for some hash function H. To activate the trigger, $\sigma = s$ would be published and listeners could verify that $H(\sigma) = p = H(s)$. This simplistic scheme has one main disadvantage: It is a fundamentally one-time construction. To prevent an attacker from re-triggering a listener a second time by replaying a valid trigger σ all listeners have to blacklist any "used" σ. Alas, this means we can only ever trigger a listener *once*. The good part is that any listener that missed this trigger can still be triggered later, but the bad part is that once s is burned we are out of options. The trivial solution to this would be to simply inform each listener with a whole list of public keys in advance. This however takes n times the amount of space for n-fold retriggerability. Luckily we can easily derive a scheme that yields n-fold retriggerability while using no more same space than the original scheme by taking some inspiration from Winternitz signatures above.

In this scheme the secret key s is still a random bit string. The public key is $p = H^n(s)$ for n-times retriggerability. The i-th time the trigger is activated, $\sigma_i = H^n - i(s)$ is published, and every listener can verify that $\sigma_{i-1} = H\left(\sigma_i\right)$ with $\sigma_0 = p$. In case a listener missed one or more previous triggers it can simply continue computing $H\left(H\left(\sigma_i\right)\right)$ and $H\left(H\left(H\left(\sigma_i\right)\right)\right)$ until either reaching the n-th recursion level (indicating an invalid signature) or finding $H^n\left(\sigma_i\right) = \sigma_j$ with $sigma_j$ being the last signature this listener recorded, or p in case there is none.

This scheme provides replay protection through listeners memorizing the last signature they activated to. Public key length is equal to the length of the hash function H used. Even for our embedded systems use case n can realistically be up to $\mathcal{O}\left(10^3\right)$, which is easily enough for our application.

Chapter 3

Practical implementation

3.1 Data collection for channel validation

To design a solid system we needed to parametrize mains frequency variations under normal conditions. To set modulation amplitude as well as parameters of our modulation scheme we need a frequency spectrum of mains frequency variations (that is $\mathcal{F}\left(f(V(t))\right)$: Taking mains frequency $f(x)$ as a variable, the frequency spectrum of that variable, as opposed to the frequency spectrum of mains voltage $V(t)$ itself).

3.1.1 Grid Frequency Estimation

In commercial power systems Phasor Measurement Units (PMUs) are used to precisely measure parameters of a mains voltage waveform. One of the parameters PMUs measure is mains frequency. PMUs are used as part of SCADA systems controlling transmission networks to characterize the operational state of the network.

From a superficial viewpoint measuring mains frequency might seem like a simple problem. Take the mains voltage waveform, measure time between two rising-edge (or falling-edge) zero-crossings and take the inverse $f = t^{-1}$. In practice, phasor measurement units are significantly more complex than this. This discrepancy is due to the unhealthy combination of both high precision and quick response that is demanded from these units. High precision is necessary since variations of mains frequency under normal operating conditions are quite small–in the range of 5 mHz to 10 mHz over short intervals of time. Relative to the nominal 50 Hz this is a derivation of less than 100 ppm Relative to the corresponding 20 ms period that means a time derivation of about 2μs from cycle to cycle. From this it is already obvious why a simplistic measurement cannot yield the required precision for manageable averaging times–we would need either a ADC sampling rate in the order of megabits or for a reconstruction through interpolated readings an impractically high ADC resolution.

Detail on the inner workings of commercial phasor measurement units is scarce but given their essential role to SCADA systems there is a large amount of academic research on such algorithms[103, 51, 9]. A popular approach to these systems is to perform a Short-Time Fourier Transform (STFT) on ADC data sampled at high sampling rate (e.g. 10 kHz) and then perform some analysis on the frequency-domain data to precisely locate the strong peak around 50 Hz. A key observation here is that FFT bin size is going to be much larger than required frequency resolution. This fundamental limitiation follows from the nyquist criterion and if we had to

process an *arbitrary* signal this would highly limit our practical measurement accuracy [1]. For this reason all approaches to mains frequency estimation are based on a model of the mains voltage waveform. Nominally, this waveform would be a perfect sine at $f = 50\,\mathrm{Hz}$. In practice it is a sine at $f \approx 50\,\mathrm{Hz}$ superimposed with some aperiodic noise (e.g. irregular spikes from inductive loads being energized) as well as harmonic distortion that is caused by grid-topologically nearby devices with power factor [2] $\cos\theta \neq 1.0$. Under a continous fourier transform over a long period the frequency spectrum of a signal distorted like this will be a low noise floor depending mainly on aperiodic noise on which a comb of harmonics as well as some sub-harmonics of $f \approx f_{\mathrm{nom}} = 50\,\mathrm{Hz}$ rides. The main peak at $f \approx f_{\mathrm{nom}}$ will be very strong with the harmonics being approximately an order of magnitude weaker in energy and the noise floor being at least another order of magnitude weaker. See figure 3.7 for a measured spectrum. This domain knowledge about the expected frequency spectrum of the signal can be employed in a number of interpolation techniques to re-construct the precise frequency of the spectrum's main component despite comparatively coarse STFT resolution and despite numerous distortions.

Published grid frequency estimation algorithms such as Narduzzi et al. [103] or Derviškadić, Romano, and Paolone [51] are rather sophisticated and use a combination of techniques to reduce numerical errors in FFT calculation and peak fitting. Given that we do not need reference standard-grade accuracy for our application we chose to start with a very basic algorithm instead. We chose to use a general approach to estimate the precise fundamental frequency of an arbitrary signal that was developed by experimental physicists at CERN and that is described by Gasior and Gonzalez [68]. This approach assumes a general sinusoidal signal superimposed with harmonics and broadband noise. Applicable to a wide spectrum of practical signal analysis tasks it is a reasonable first-degree approximation of the much more sophisticated estimation algorithms developed specifically for power systems. Some algorithms have components such

[1] Some software packages providing FFT or STFT primitives such as scipy[133] allow the user to super-sample FFT output by specifying an FFT width larger than input data length, padding the input data with zeros on both sides. Note that in line with Nyquist this *does not* actually provide finer output resolution but instead just amounts to an interpolation between output bins. Depending on the downstream analysis algorithm it may still be sensible to use this property of the DFT for interpolation, but in general it will be computationally expensive compared to other interpolation methods and in any case it will not yield any better frequency resolution aside from a hypothetical numerical advantage[69].

[2] Power factor is a power engineering term that is used to describe how close the current waveform of a load is to that of a purely resistive load. Given sinusoidal input voltage $V(t) = V_{\mathrm{pk}} \sin(\omega_{\mathrm{nom}} t)$ with $\omega_{\mathrm{nom}} = 2\pi f_{\mathrm{nom}} = 2\pi \cdot 50\,\mathrm{Hz}$ being the nominal angular frequency, the current waveform of a resistor with resistance $R\,[\Omega]$ according to Ohm's law would be $I(t) = \frac{V(t)}{R} = \frac{1}{R} V_{\mathrm{pk}} \sin(\omega_{\mathrm{nom}} t)$. In this case voltage and current are perfectly in phase, i.e. the current at time t is linear in voltage at constant factor $\frac{1}{R}$.

In contrast to this idealized scenario reality provides us with two common issues: One, the load may be reactive. This means its current waveform is an ideal sinusoid, but there is a phase difference between mains voltage and load current like so: $I(t) = \frac{V(t)}{R} = \frac{1}{|Z|} V_{\mathrm{pk}} \sin(\omega_{\mathrm{nom}} t + \varphi)$ Z would be the load's complex impedance combining inductive, capacitive and resistive components and φ the phase difference between the resulting current waveform and the mains voltage waveform. A common case of such loads are motors and the inductive ballasts in old fluorescent lighting fixtures.

The second potential issue are loads with non-sinusoidal current waveform. There are many classes of these but the most common one are switching-mode power supplies. Most SMPS for modern electronic devices have an input stage consisting of a bridge rectifier followed by a capacitor that provide high-voltage DC power to the following switch-mode convert circuit. This rectifier-capacitor input stage under normal load draws a high current only at the very peak of the input voltage sinusoid and draws almost zero current for most of the period.

These two cases are measured by *displacement power factor* and *distortion power factor* that when combined yield the overall true power factor. The power factor is a key quantity in the design and operation of the power grid since a high power factor (close to 1.0 or an in-phase sinusoidal current waveform) yields lowest transmission and generation losses.

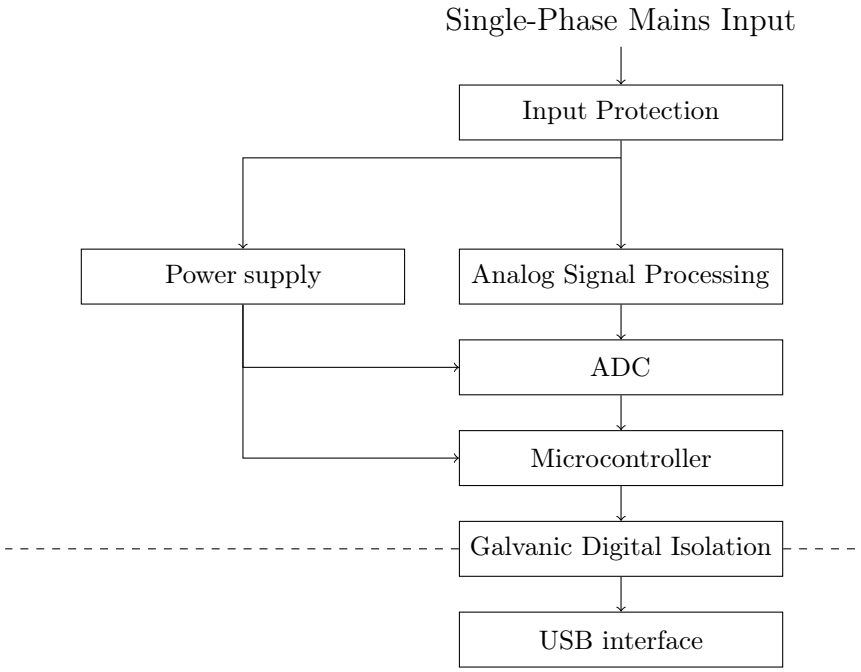


Figure 3.1: Frequency sensor hardware diagram

as kalman filters[103] that require a phyiscal model. As a general algorithm from Gasior and Gonzalez [68] does not require this kind of application-specific tuning, eliminating one source of error.

The Gasior and Gonzalez [68] algorithm passes the windowed input signal through a DFT, then interpolates the signal's fundamental frequency by fitting a wavelet such as a gaussian to the largest peak in the DFT results. The bias parameter of this curve fit is an accurate estimation of the signal's fundamental frequency. This algorithm is similar to the simpler interpolated DFT algorithm used as a reference in much of the synchrophasor estimation literature[11]. The three-term variant of the maximum sidelobe decay window often used there is a blackman window with parameter $\alpha = \frac{1}{4}$. Analysis has shown[9] that the interpolated DFT algorithm is worse than algorithms involving more complex models under some conditions but that there is *no free lunch* meaning that more complex perform worse when the input signal deviates from their models.

3.1.2 Frequency sensor hardware design

Our safety reset controller will have to measure mains frequency to later demodulate a reset signal transmitted through it. Since we have decided to do our own frequency measurement system here we can use this frequency measurement setup as a prototype for the frequency measurement subcomponent of the demodulation system we will later develop. Since we do not plan to do a large-scale field deployment of our measurement setup we can keep the hardware implementation simple by moving most of the signal processing to a regular computer and concentrating our hardware efforts on raw signal capture.

An overall block diagram of our system is shown in Figure 3.1. The mircrocontroller we chose is an `STM32F030F4P6` ARM Cortex-M0 microcontroller made by ST Microelectronics. The ADC in Figure 3.1 in our design is the integrated 12-bit ADC of this microcontroller, which is

sufficient for our purposes. The USB interface is a simple USB to serial converter IC (`CH340G`) and the galvanic digital isolation is accomplished with a pair of high-speed optocouplers on its `RX` and `TX` lines. The analog signal processing is a simple voltage divider using high-power resistors to get the required creepage along with some high-frequency filter capacitors and an op-amp buffer. The power supply is an off-the-shelf mains-input power module. The system is implemented on a single two-layer PCB that is housed in an off-the-shelf industrial plastic case fitted with a printed label and a few status lights on its front.

3.1.3 Clock accuracy considerations

Our measurement hardware will sample line voltage at some sampling rate f_S, e.g. 1 kHz. All downstream processsing is limited in accuracy by the accuracy of f_S[3]. We generate our sampling clock in hardware by clocking the ADC from one of the microcontroller's timer blocks clocked from the microcontroller's system clock. This means our ADC's sampling window will be synchronized cycle-accurate to the microcontroller's system clock.

Our downstream measurement of mains frequency by nature is relative to our sampling frequency f_S. In the setup described above this means we have to make sure our system clock is fairly stable. A frequency derivation of 1 ppm in our system clock causes a proportional grid frequency measurement error of $\Delta f = f_{\text{nom}} \cdot 10^{-6} = 50\,\mu\text{Hz}$. In a worst-case where our system is clocked from a particularly bad crystal that exhibits 100 ppm of instabilities over our measurement period we end up with an error of 5 mHz. This is well within our target measurement range, so we need a more stable clock source. Ideally we want to avoid writing our own clock conditioning code where we try to change an oscillators operating frequency to match some reference. Clock conditioning algorithms are highly complex and in our case post-processing of measurement data and simply adding and offset is simpler and less error-prone.

Our solution to these problems is to use a crystal oven[4]as our main system clock source. Crystal ovens are expensive compared to ordinary crystal oscillators. Since any crystal oven will be much more accurate than a standard room-temperature crystal we chose to reduce cost by using one recycled from old telecommunications equipment.

To verify clock accuracy we routed an externally accessible SMA connector to a microcontroller pin that is routed to one of the microcontroller's timer inputs. By connecting a GPS 1pps signal to this pin and measuring its period we can calculate our system's Allan variance[5], thereby measuring both clock stability and clock accuracy. We ran a 4 hour test of our frequency sensor that generated the histogram shown in figure 3.2. These results show that while we get a systematic error of about 10 ppm due to manufacturing tolerances the random error at less than 10 ppb is smaller than that of a room-temperature crystal oscillator by 3-4 orders of magnitude. Since we are interested in grid frequency variations over time but not in the absolute value of grid frequency the systematic error is of no consequence to us. The random error at 3.66 ppb corresponds to a frequency measurement error of about 0.2 µHz, well below what we can achieve at reasonable sampling rates and ADC resolution.

[3]We are not considering the effects of clock jitter. We are highly oversampling the signal and the FFT done in our downstream processing will eliminate small jitter effects leaving only frequency stability to worry about.

[4]A crystal oven is a crystal oscillator thermally coupled closely to a heater and temperature sensor and enclosed in a thermally isolated case. The heater is controlled to hold the crystal oscillator at a near-constant temperature some few ten degrees above ambient. Any ambient temperature variations will be absorbed by the temperature control. This yields a crystal frequency that is almost completely unaffected by ambient temperature variations below the oven temperature and whose main remaining instability is aging.

[5]Allan variance is a measure of frequency stability between two clocks.

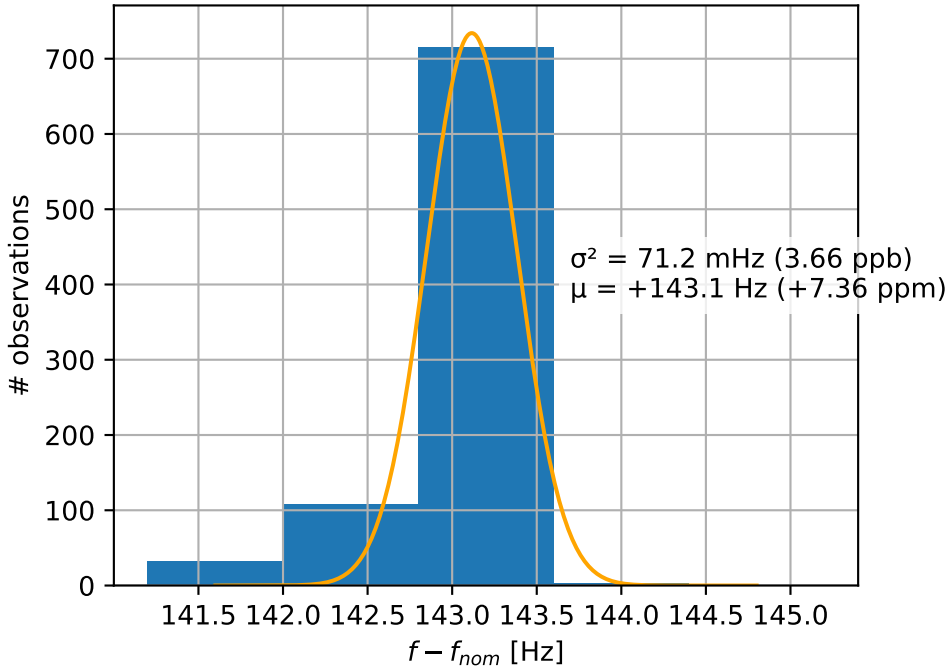


Figure 3.2: OCXO Frequency derivation from nominal 19.440 MHz measured against GPS 1pps

3.1.4 Firmware implementation

The firmware uses one of the microcontroller's timers clocked from an external crystal oscillator to produce an 1 ms tick that the internal ADC is triggered from for a sample rate of 1 ksps. Higher sample rates would be possible but reliable data transmission over the opto-isolated serial interface might prove challenging and 1 ksps corresponds to 20 samples per cycle at f_{nominal}. This is 10× nyquist and should be plenty for accurate measurements.

The ADC measurements are read using DMA and written into a circular buffer. Using some DMA controller features this circular buffer is split in back and front halves with one being written to and the other being read at the same time. Buffer contents are moved from the ADC DMA buffer into a packet-based reliable UART interface as they come in. The UART packet interface keeps two ringbuffers: One byte-based ringbuffer for transmission data and one ringbuffer pointer structure that keeps track of ADC data packet boundaries in the byte-based ringbuffer. Every time a chunk of data is available from the ADC the data is framed into the byte-based ringbuffer and the packet boundaries are logged in the packet pointer ringbuffer. If the UART transmitter is idle at this time a DMA-backed transmission of the oldest packet in the packet ringbuffer is triggered at this point. Data is framed using Consistent Overhead Byte Stuffing (COBS)[6][40] along with a CRC-32 checksum for error checking. When the host receives a new packet with a valid checksum it returns an acknowledgement packet to the sensor.

[6]COBS is a framing technique that allows encoding n bytes of arbitray data into exactly $n + 1$ bytes with no embedded 0-bytes that can then be delimited using 0-bytes. COBS is simple to implement and allows both one-pass decoding and encoding. The encoder either needs to be able to read up to 256 B ahead or needs a buffer of 256 B. COBS is very robust in that it allows self-synchronization. At any point a receiver can reliably synchronize itself against a COBS data stream by waiting for the next 0-byte. The constant overhead allows precise bandwidth and buffer planning and provides constant, good efficiency close to the theoretical maximum.

When the sensor receives the acknowledgement, the acknowledged packet is dropped from the transmission packet ringbuffer. When the host detects an incorrect checksum it simply stays quiet and waits for the sensor to resume with retransmission when the next ADC buffer has been received.

The serial interface logic presents most of the complexity of the sensor firmware. This complexity is necessary since we need reliable, error-checked transmission to the host. Though rare, bit errors on a serial interface do happen and data corruption is unacceptable. The packet-layer queueing on the sensor is necessary since the host is not a realtime system and unpredictable latency spikes of several hundred milliseconds are possible.

The host in our recording setup is a Raspberry Pi 3 model B running a Python script. The Python script handles serial communication and logs data and errors into an SQLite database file. SQLite has been chosen for its simple yet flexible interface and its good tolerance of system resets due to unexpected power loss. Overall our setup performed adequately with IO contention on the raspberry PI/linux side causing only 16 skipped sample packets over a 68-hour recording span.

3.1.5 Frequency sensor measurement results

Captured raw waveform data has been processed in the Jupyter Lab environment[84] and grid frequency estimates are extracted as described in sec. 3.1.1 using the Gasior and Gonzalez [68] technique. Appendix A.1 contains the Jupyter notebook we used for frequency measurement. In Figure 3.3 we fed back to the frequency estimator its own output giving us an indication of its numerical performance. The result was 1.3 mHz of RMS noise over a 3600 s simulation time. This indicates performance is good enough for our purposes. In addition to this we validated our algorithm's performance by applying it to the test waveforms from Wright [136]. In this test we got errors of 4.4 mHz for the *noise* test waveform, 0.027 mHz for the *interharmonics* test waveform and 46 mHz for the *amplitude and phase step* test waveform. Full results can be found in Figure 3.4.

Figures 3.5 and 3.6 show our measurement results over a 24-hour and a 2-hour window respectively.

3.2 Channel simulation and parameter validation

To validate all layers of our communication stack from modulation scheme to cryptography we built a prototype implementation in python. Implementing all components in a high-level language builds up familiartiy with the concepts while taking away much of the implementation complexity. For our demonstrator we will not be able to use python since our target platform is a cheap low-end microcontroller. Our demonstrator firmware will have to be written in a low-level language such as C or rust. For prototyping these languages lack flexibility compared to python.

To validate our modulation scheme we first performed a series of simulations on our python demodulator prototype implementation. To simulate a modulated grid frequency signal we added noise to a synthetic modulation signal. For most simulations we used measured frequency data gathered with our frequency sensor. We only have a limited amount of capture data. Re-using segements of this data as background noise in multiple simulation runs could hypothetically lead to our simulation results depending on individual features of this particular capture that would be common between all runs. To estimate the impact of this problem we re-ran some of

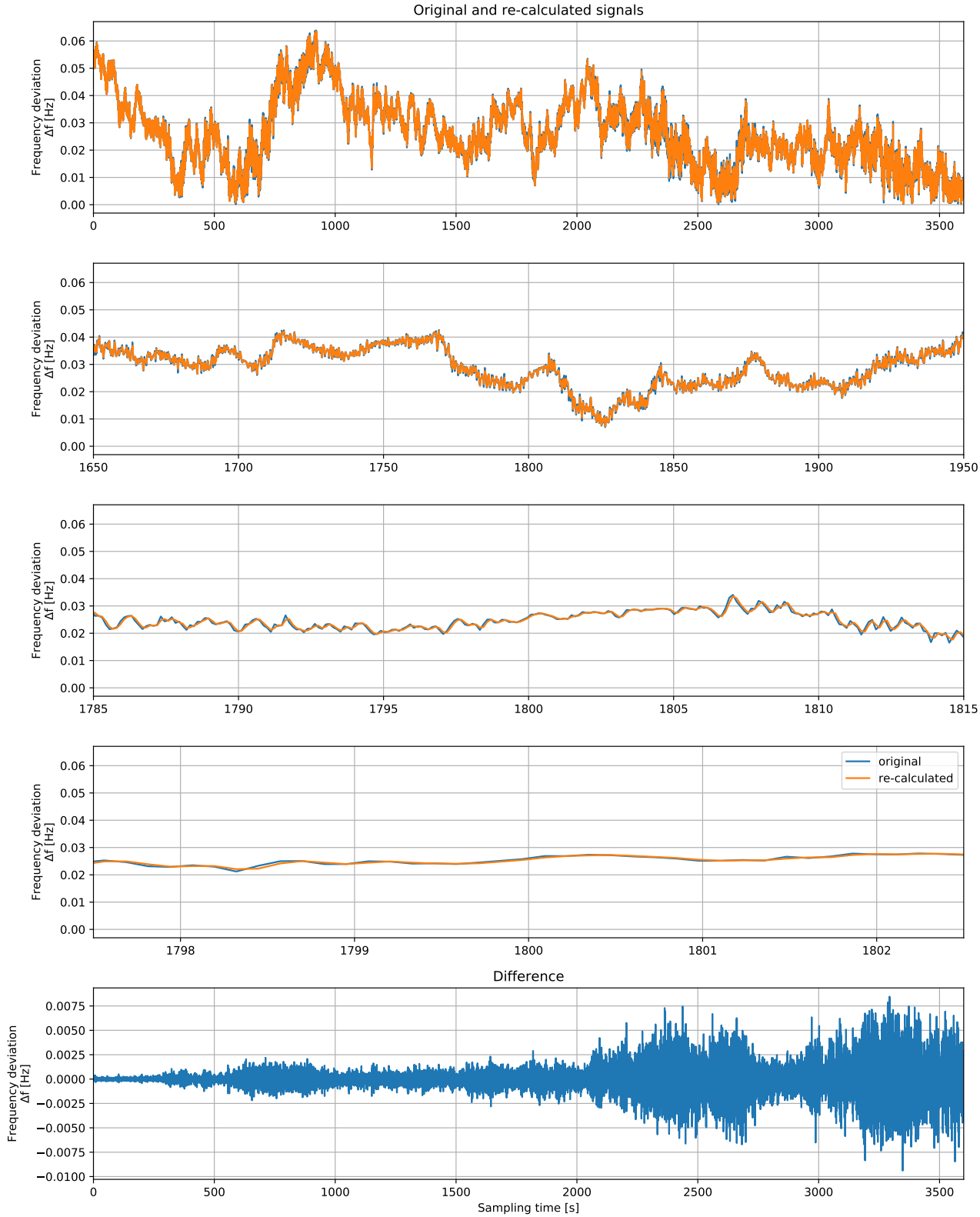


Figure 3.3: The frequency estimation algorithm applied to a synthetic noise-less mains waveform generated from its own output. This feedback simulation gives an indication of numerical errors in our estimation algorithm. The top four graphs show a comparison of the original trace (blue) and the re-calculated trace (orange). The bottom trace shows the difference between the two. As we can tell both traces agree very well with an overall RMS deviation of about 1.3 mHz. The bottom trace shows deviation growing over time. This is very likely an effect of numerical errors in our ad-hoc waveform generator.

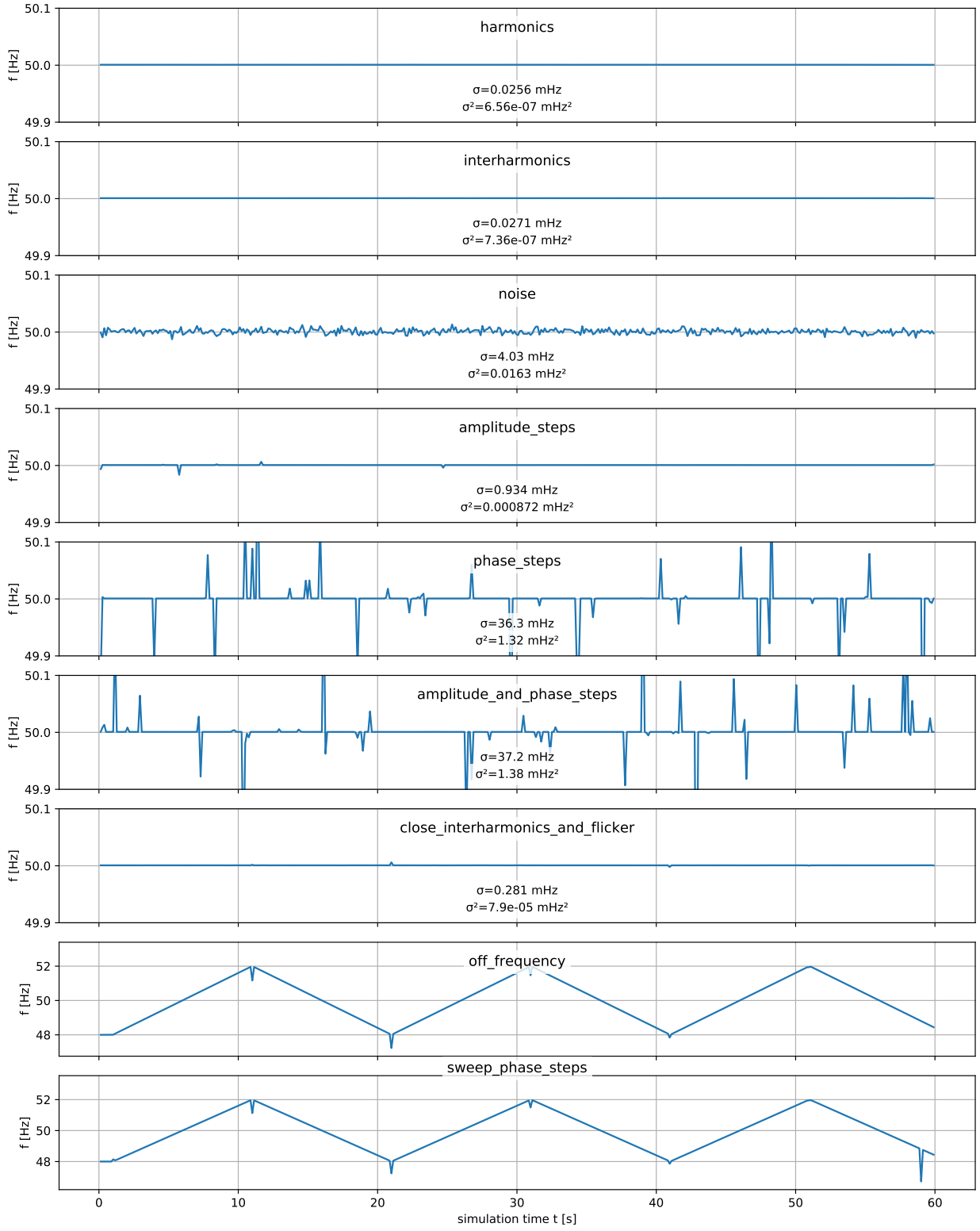


Figure 3.4: Performance of our frequency estimation algorithm against the test suite specified in Wright [136]. Shown are standard deviation and variance measurements as well as time-domain traces of differences.

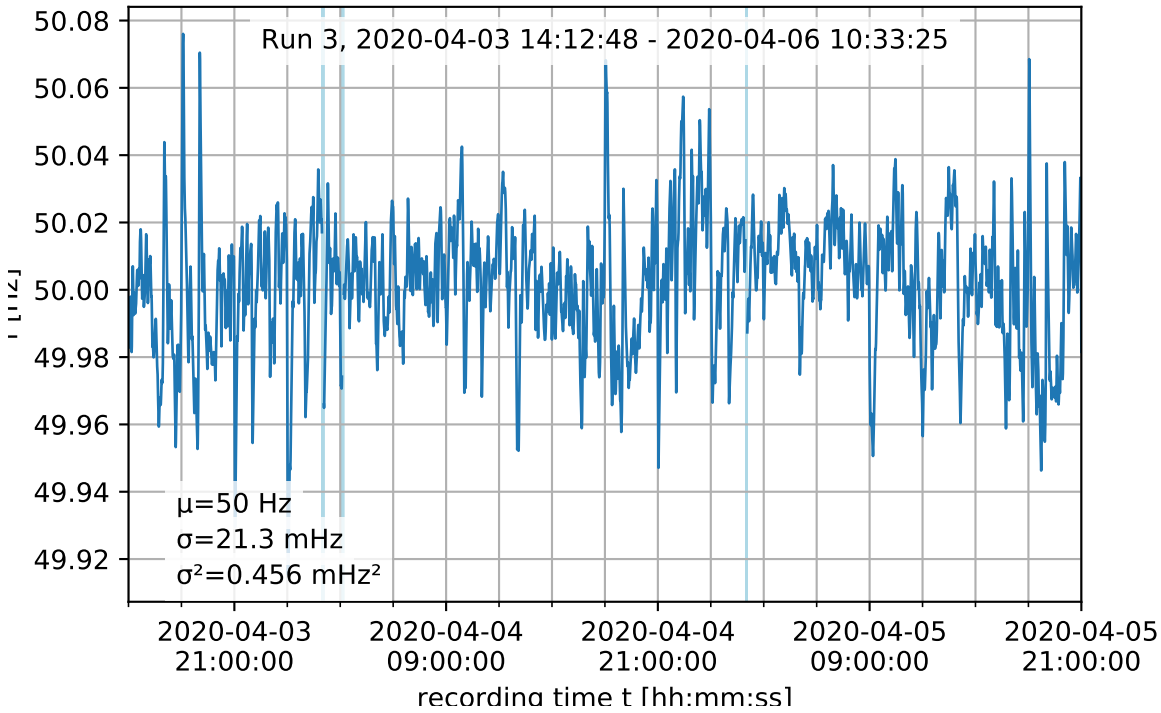
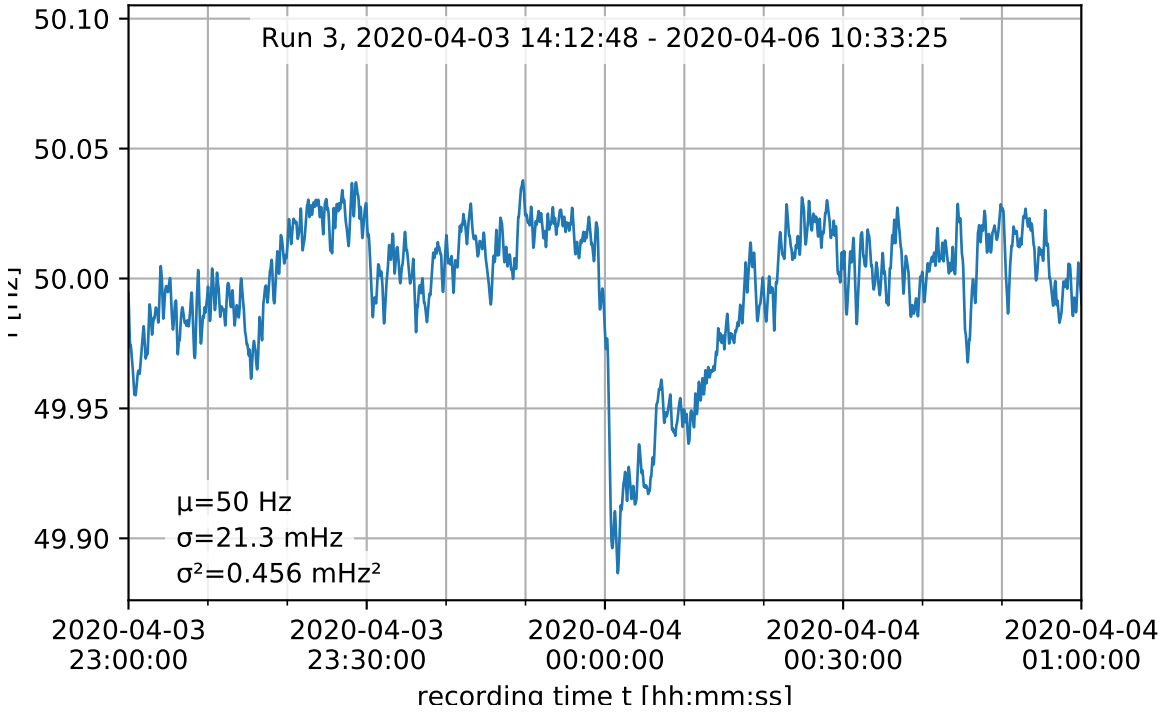
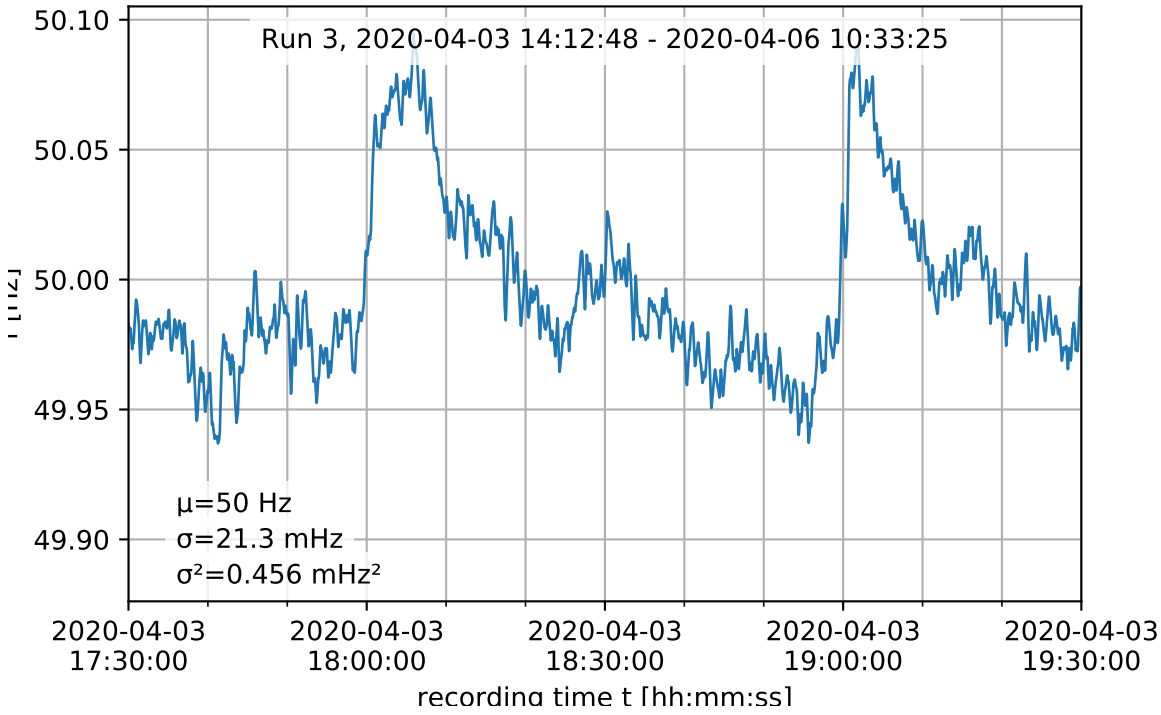


Figure 3.5: Trace of grid frequency over a 24 hour window. One clearly visible feature are large positive and negative transients at full hours. Times shown are UTC. Note that the european continental synchronous area that this sensor is placed in covers several time zones which may result in images of daily load peaks appearing in 1 hour intervals. Figure 3.6 contains two magnified intervals from this plot.



(a) A 2 hour window around 00:00 UTC.



(b) A 2 hour window around 18:30 UTC.

Figure 3.6: Two magnified 2 hour windows of the trace from Figure 3.5.

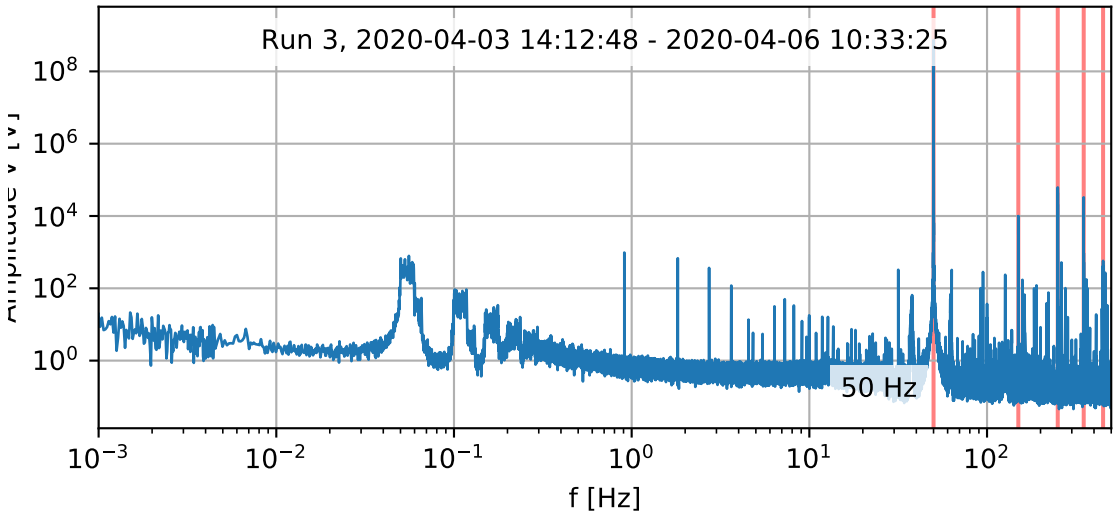


Figure 3.7: Power spectral density of the mains voltage trace in Figure 3.5. Data was captured using our frequency measurement sensor (3.1.2) and FFT'ed after applying a blackman window. Vertical lines indicate 50 Hz and odd harmonics. We can see the expected peak at 50 Hz along with smaller peaks at odd harmonics. We can also see a number of spurious tones both between harmonics and at low frequencies, as well as some bands containing high noise energy around 0.1 Hz. This graph demonstrates a high signal-to-noise ratio that is not very demanding on our frequency estimation algorithm.

our simulations with artificial random noise synthesized with a power spectral density matching that of our capture. To do this, we first measured our capture's PSD, then fitted a low-resolution spline to the PSD curve in log-log coördinates. We then generated white noise, multiplied the resampled spline with the DFT of the synthetic noise and performed an iDFT on the result. The resulting time-domain signal is our synthetic grid frequency data. Figure 3.8 shows the PSD of our measured grid frequency signal. The red line indicates the low-resolution log-log spline interpolation used for shaping our artificial noise. Figure 3.9 shows the PSD of our simulated signal overlayed with the same spline as a red line and shows time-domain traces of both simulated (blue) and reference signals (orange) at various time scales. Visually both signals look very similar, suggesting we have found a good synthetic approximation of our measurements.

In our simulations, we manipulated four main variables of our modulation scheme and demodulation algorithm and observed their impact on symbol error rate (SER):

Modulation amplitude. Higher amplitude should correspond to a lower SER.

Modulation bit count. Higher bit count n means longer transmissions but yields higher theoretical decoding gain, and should increase demodulator sensitivity. Ultimately, we want to find a sweet spot of manageable transmission length at good demodulator sensitivity.

Decimation. or DSSS chip duration. The chip time determines where in the grid frequency spectrum (Figure 3.8 our modulated signal is located. Given our noise spectrum (Figure 3.8) lower chip durations (shifting our signal upwards in the spectrum) should yield lower in-band background noise which should correspond to lower symbol error rates.

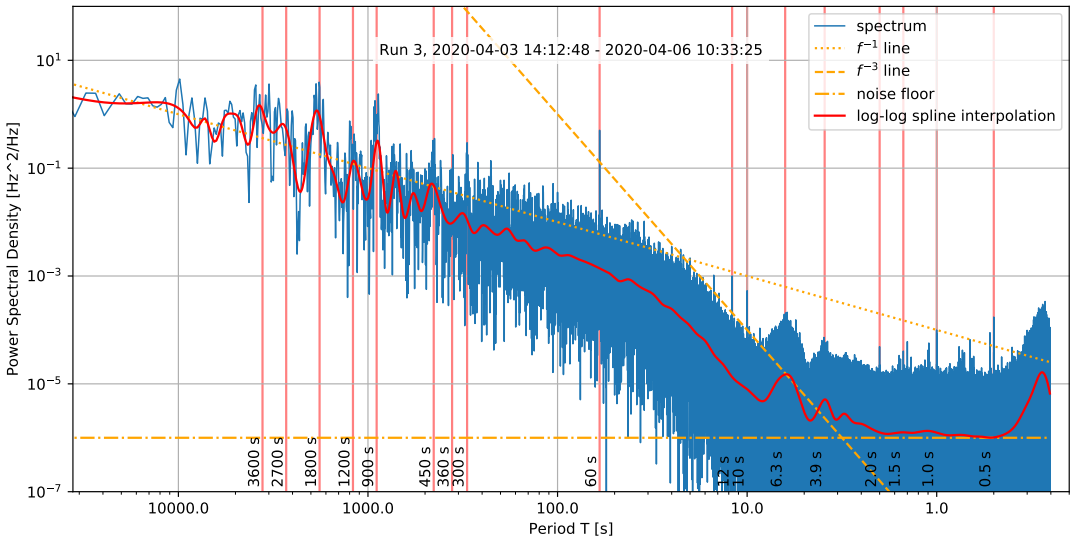


Figure 3.8: Power spectral density of the 24 hour grid frequency trace in Figure 3.5 with some notable peaks annotated with the corresponding period in seconds. The $\frac{1}{f}$ line indicates a pink noise spectrum. Around a period of 20 s the PSD starts to fall off at about $\frac{1}{f^3}$ until we can make out some bumps at periods around 2 and 3 s. Starting at at around 1 Hz we can see a white noise floor in the order of $\mu\text{Hz}^2/\text{Hz}$.

Demodulation correlator peak threshold factor. The first step of our prototype demodulation algorithm is to calculate the correlation between all $2^n + 1$ Gold sequences and to identify peaks corresponding to the input data containing a correctly aligned Gold sequence. The threshold factor is a factor peaks of what magnitude compared to baseline noise levels are considered in the following maximum likelihood estimation (MLE) decoding.

Our results indicate that symbol error rate is a good proxy of demodulation performance. With decreasing signal-to-noise ratio, margins in various parts of the demodulator decrease which statistically leads to an increased symbol error rate. Our simulations yield smooth, reproducible SER curves with adequately low error bounds. This shows SER is related monotonically to the signal-to-noise margins inside our demodulator prototype.

3.2.1 Sensitivity as a function of sequency length

A basic parameter of our DSSS modulation is the length of the Gold codes used. The length of a Gold code is exponential in the code's bit count. Figure 3.10 shows a plot of the symbol error rate of our demodulator prototype depending on amplitude for each of five, six, seven and eigth-bit Gold sequences. In regions where symbol error rate is between 0 and 1 we can see the expected dependency that a $n + 1$ bit Gold sequence at roughly twice the length yields roughly one half the SER. We can also observe a saturation effect: At low amplitudes, increasing the correlation length does not seem to yield much of a benefit in SER anymore. In particular there seems to be a level of about 2.5 mHz signal amplitude where even with asymptotically infinite sequence length our demodulator would still not be able to produce a good demodulation. This

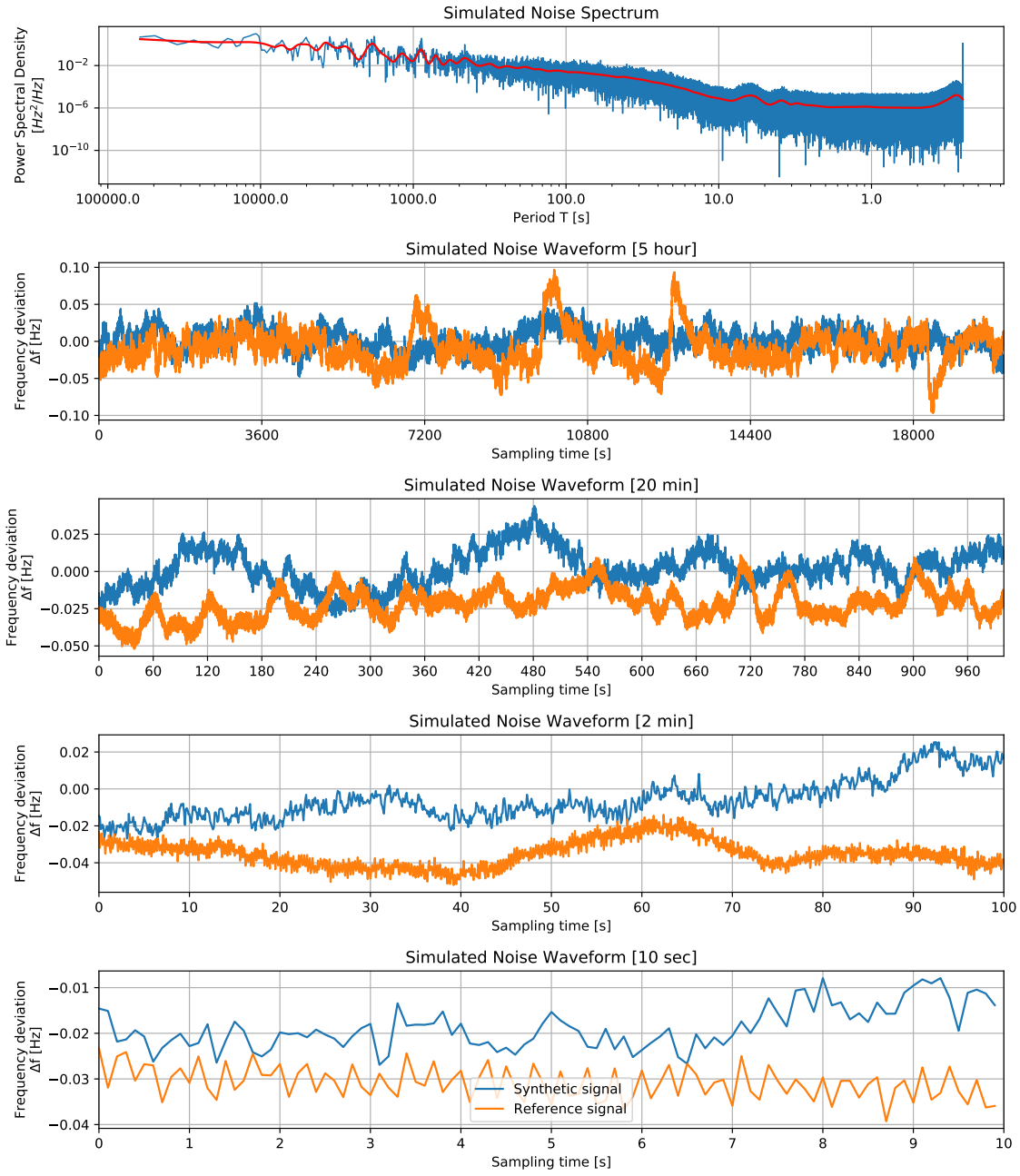


Figure 3.9: Synthetic grid frequency in comparison with measured data. The topmost graph shows the synthetic spectrum compared to the spline approximation of the measured spectrum (red line). The other graphs show time-domain synthetic data (blue) in comparison with measured data (orange).

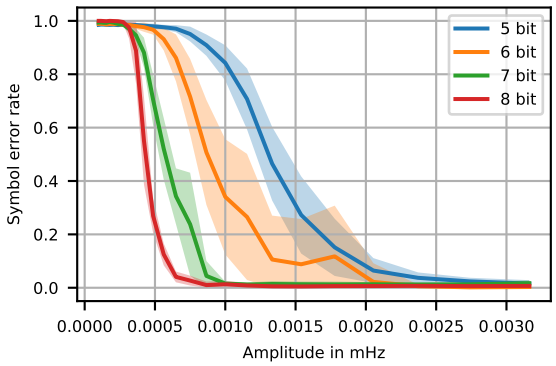


Figure 3.10: Symbol Error Rate (SER) as a function of transmission amplitude. The line represents the mean of several measurements for each parameter set. The shaded areas indicate one standard deviation from the mean. Background noise for each trial is a random segment of measured grid frequency. Background noise amplitude is the same for all trials. Shown are four traces for four different DSSS sequence lengths. Using a 5-bit gold code, one DSSS symbol measures 31 chips. 6 bit per symbol are 63 chips, 7 bit are 127 chips and 8 bit 255 chips. This simulation uses a decimation of 10, which corresponds to an 1s chip length at our 10Hz grid frequency sampling rate. At 5 bit per symbol, one symbol takes 31s and one bit takes 6.2s amortized. At 8 bit one symbol takes 255s = 4min15s and one bit takes 31.9s amortized. Here, slower transmission speed buys coding gain. All else being the same this allows for a decrease in transmission power.

is likely due to numerical errors in our demodulator. Since Gold codes of more than 7 bit would yield unacceptably long transmission times this does not pose a problem in practice.

Figure 3.11 for each bit count shows the minimum signal amplitude where our demodulator crossed below SER = 0.5. If we have sufficient transmitter power to allocate selecting either a 5 bit or a 6 bit gold code looks to yield good enough performance at manageable data rates.

3.2.2 Sensitivity versus peak detection threshold factor

One of the high-level parameters of our demodulation algorithm is the *threshold factor*. This parameter is an implementation detail specific to our algorithm and not general to all possible DSSS demodulation algorithms. After correlating the input signal against the template Gold sequences our algorithm runs a single-channel discrete wavelet transform (DWT) on the correlator output to better discriminate peaks from background noise. The output of this DWT is then normalized against a running average and then fed into a simple threshold detector. The threshold of this detector is our threshold factor. This threshold is the ratio that a correlation peak after DWT has to stand out from long-term average background noise to be considered a peak.

The threshold factor is an empirically-determined parameter Low threshold factors yield many false positives that in the extreme ultimately overload our MLE estimator's capacity to discard them. Moderate numbers of false positive do not pose much of a challenge to our MLE since these spurious peaks have a random time distribution and are easily discarded by our MLE's symbol chain detection. High threshold factors lead the algorithm to completely ignore some valid peaks. To some degree this can be compensated by our later interpolation step for missing peaks but in the extreme will also break demodulation. In our simulations good values lie in the range from 4.0 to 5.5.

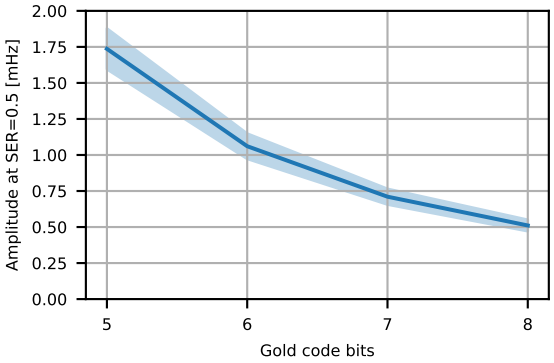


Figure 3.11: Amplitude at a SER of 0.5 in mHz depending on symbol length. Here we can observe an increase of sensitivity with increasing symbol length, but we can clearly see diminishing returns above 6 bit (63 chips). Considering that each bit roughly doubles overall transmission time for a given data length it seems lower bit counts are preferrable if the necessary transmitter power can be realized.

Figure 3.12 contains plots of demodulator sensitivity like the one in Figure 3.10. This time there is one color-coded trace for each threshold factor between 1.5 and 10.0 in steps of 0.5. We can see a clear dependency of demodulation performance from trheshold factor with both very low and very high values breaking the demodulator. The "runaway" traces that we can see at low threshold factors are artifacts of an implementation issue with our prototype code. We later fixed this issue in the demonstrator firmware implementation in Section 3.3.2. For comparison purposes this issue do not matter.

If we again look at the intercept points where the amplitude traces cross SER = 0.5 in these graphs we get the plots in Figure 3.13. From this we can conclude that the range between 4.0 and 5.0 will yield adequate threshold factors for our use case.

3.2.3 Chip duration and bandwidth

A parameter of any DSSS system is the frequency band used for transmission. Instead of specifying absolute frequencies in our simulations we expressed DSSS bandwidth through chip duration and Gold sequence length. In our prototype, chip duration is specified in grid frequency sampling periods to ease implementation without loss of generalization.

Figure 3.14 shows the dependence of symbol error rate at a fixed good threshold factor from chip duration. The color bars indicate both chip duration translated to seconds real-time and the resulting symbol duration at the given Gold code length. In the lower graphs we show the trace of ampltude at SER = 0.5 over chip duration like we did in Figure 3.13 for threshold facotr. In both graphs we can just about see an optimum for very short chips with a decrease of sensitivity for long chips. This effect is due to longer chips moving the signal band into noisier spectral regions (cf. Figure 3.8).

In the previous graphs we have used random clips of measured grid frequency noise as noise in our simulations. Comparing between a simulation using measured noise and synthetic noise generated as we outlined in the beginning of Section we get the plots in Figure 3.15. We can see that while not perfect our simulated noise is an adequate approximation of reality: Our prototype demodulator shows no significant difference in behavior between measured and simulated noise. Simulated noise causes slightly worse performance for long chips. Overall the results for both

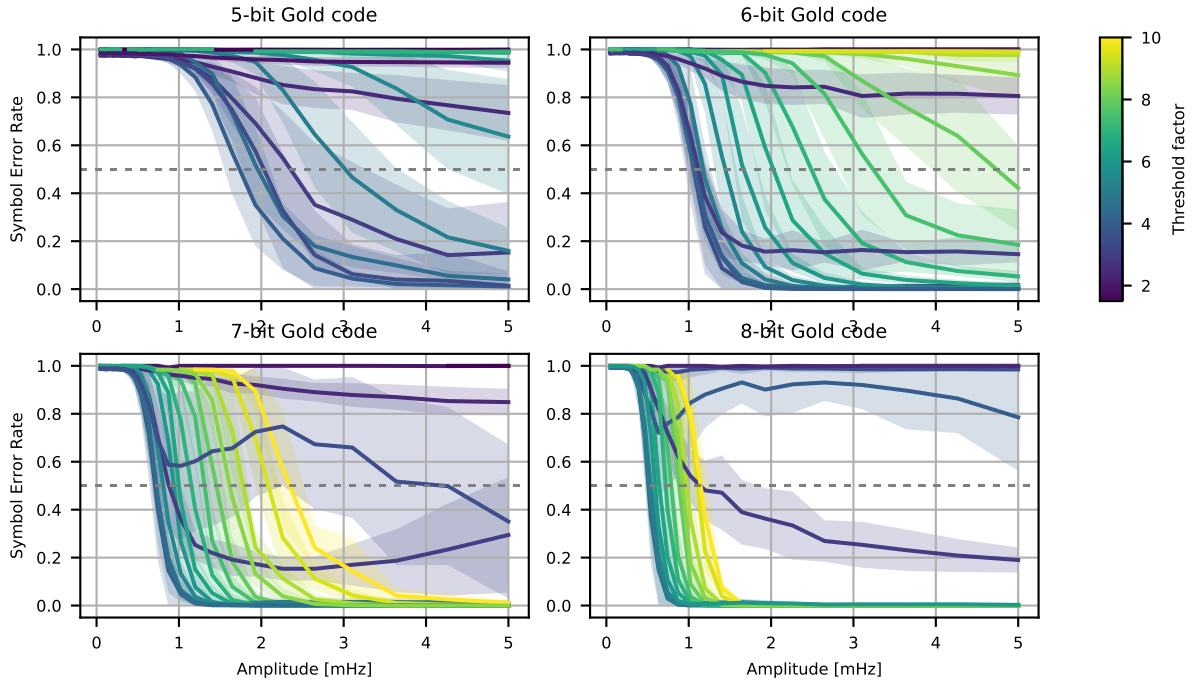


Figure 3.12: SER vs. amplitude graph similar to Figure 3.10 with one color-coded traces for threshold factors between 1.5 and 10.0. Each graph shows traces for a single DSSS symbol length.

are very close in absolute value.

3.3 Implementation of a demonstrator unit

To demonstrate the viability of our reset architecture we decided to implement a demonstrator system. In this demonstrator we use JTAG to reset part of a commodity smart meter from an externally-connected reset controller. The reset controller receives its commands over the grid frequency modulation system we outlined in this thesis. To keep implementation cost low the reset controller is fed a simulation of a modulated grid frequency signal through a standard 3.5 mm audio jack[7]. Measurement of actual grid frequency instead would simply require a voltage divider and depending on the setup an analog optoisolator.

3.3.1 Selecting a smart meter for demonstration purposes

For our demonstrator to make sense we wanted to select a realistic reset target. In Germany where this thesis was written a standards-compliant setup would consist of a fairly dumb smart meter and a smart meter gateway (SMGW) containing all of the complex bidirectional protocol logic such as wireless or landline IP connectivity. The realistic target for a setup in this architecture would be the components of an SMGW such as its communications modem or

[7]By generously cutting two PCB traces the meter we chose to use can be easily modified to provide strong galvanic separation between grid and main application microcontroller. With this modification we have to supply power to its main application MCU externally along with the JTAG interface.

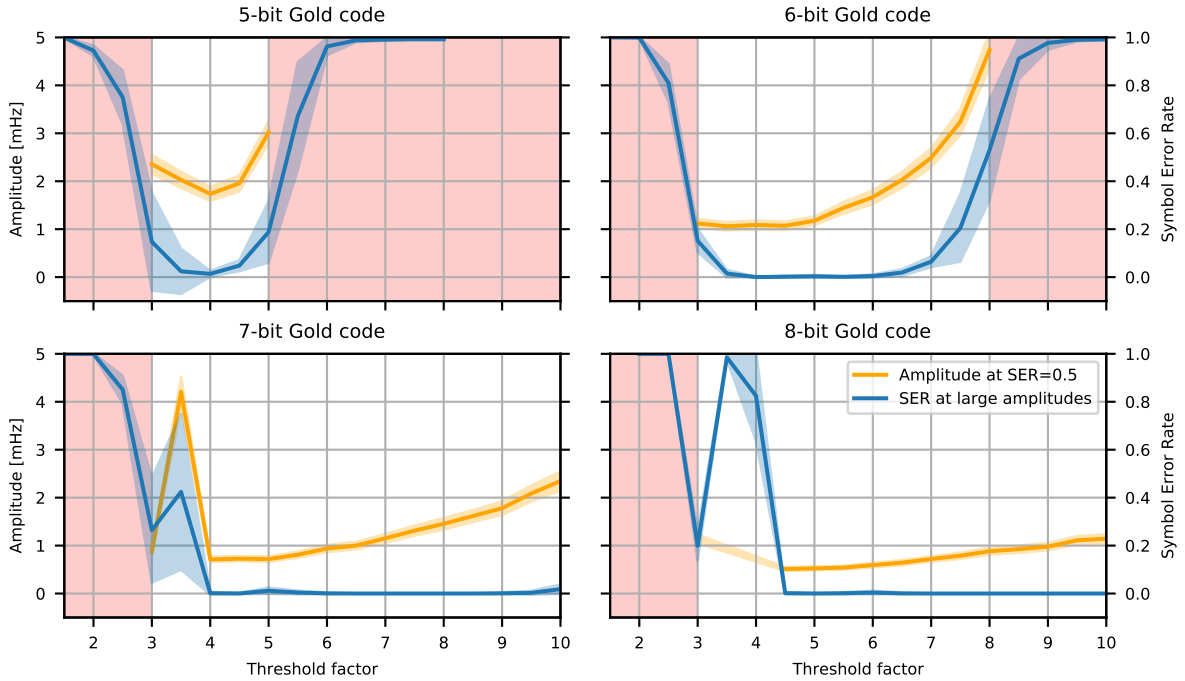
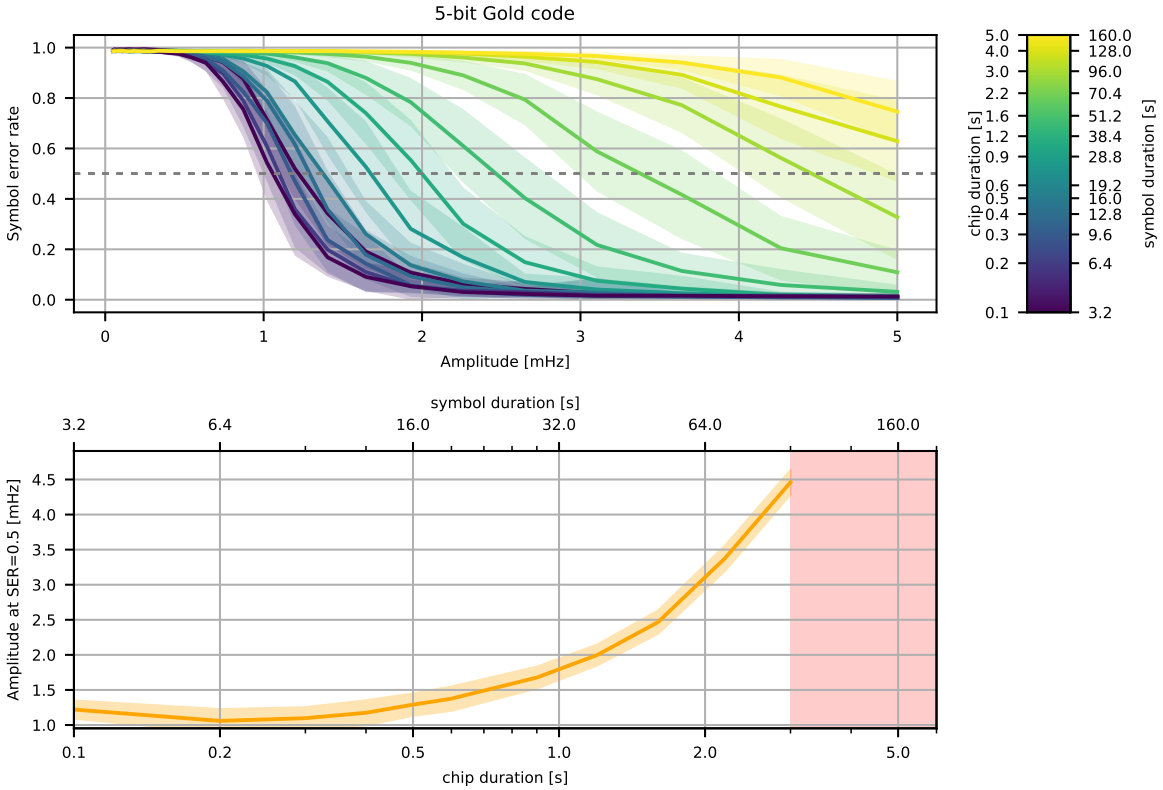
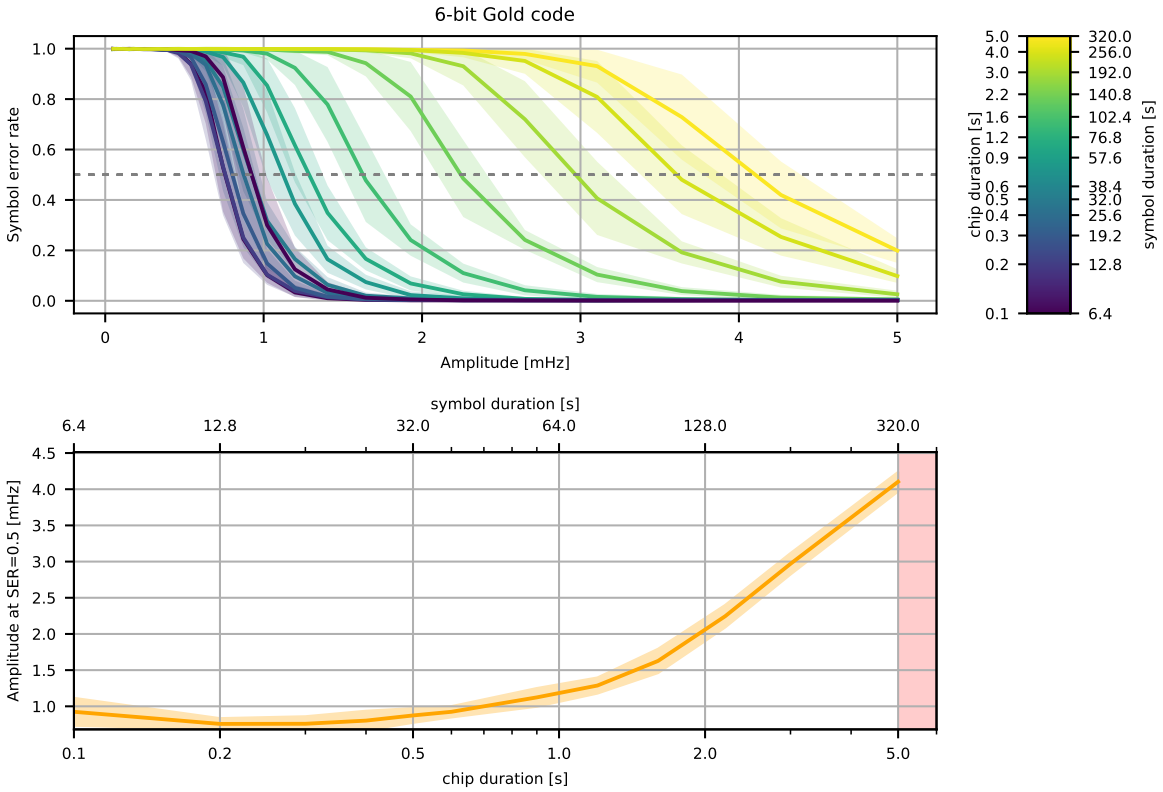


Figure 3.13: Graphs of amplitude at $SER = 0.5$ for each symbol length as well as asymptotic SER for large amplitudes. Areas shaded red indicate that $SER = 0.5$ was not reached for any amplitude in the simulated range. The bumps in the 7 bit and 8 bit graphs are due to the convergence problem we identified above and do not exist in our demonstrator implementation. We see that smaller symbol lengths favor lower threshold factors, and that optimal threshold factors for all symbol lengths are between 4.0 and 5.0.

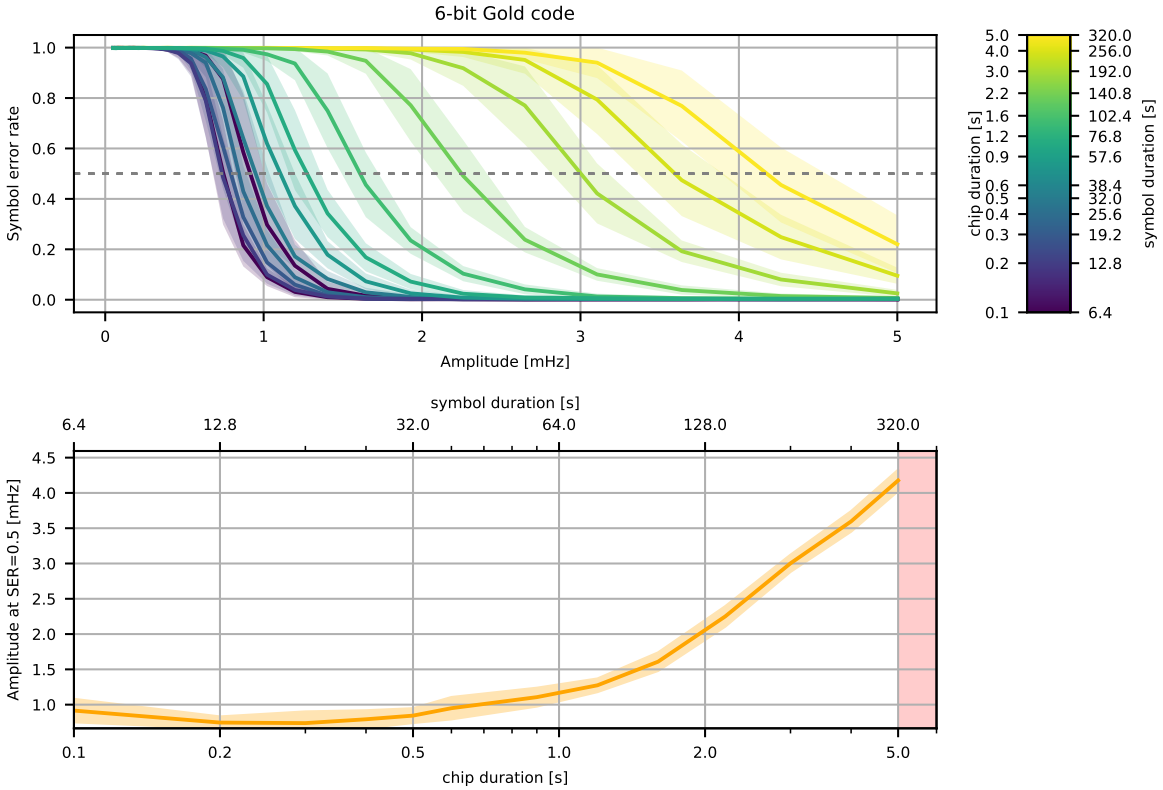


(a) 5 bit Gold code

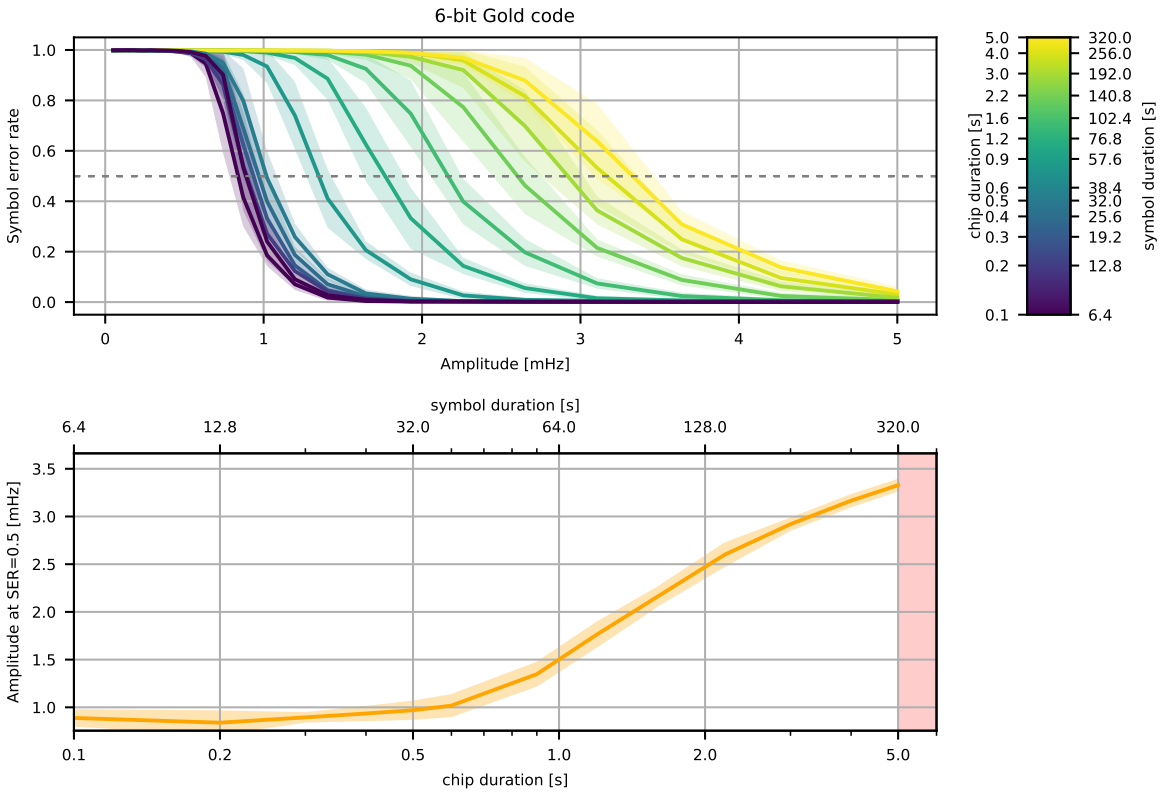


(b) 6 bit Gold code

Figure 3.14: Dependence of demodulator sensitivity on DSSS chip duration. Due to computational constraints this simulation is limited to 5 bit and 6 bit DSSS sequences. There is a clearly visible sensitivity maximum at fairly short chip lengths around 0.2s. Short chip durations shift the entire transmission band up in frequency. In Figure 3.8 we can see that noise energy is mostly concentrated at lower frequencies, so shifting our signal up in frequency will reduce the amount of noise the decoder sees behind the correlator by shifting the band of interest into a lower-noise spectral region. For a practical implementation chip duration is limited by physical factors such as the maximum modulation slew rate ($\frac{\mathrm{d}P}{\mathrm{d}t}$), the maximum Rate-Of-Change-Of-Frequency (ROCOF, $\frac{\mathrm{d}f}{\mathrm{d}t}$) the grid can tolerate and possible inertial effects limiting response of frequency to load changes at certain load levels.



(a) Simulation using baseline frequency data from actual measurements.



(b) Simulation using synthetic frequency data.

Figure 3.15: Chip duration/sensitivity simulation results like in Figure 3.14 compared between a simulation using measured frequency data like previous graphs and one using artificially generated noise. There is little visible difference indicating that we have found a good model of reality in our noise synthesizer, but also that real grid frequency behaves like a frequency-shaped gaussian noise process.

main application processor. In the German architecture the smart meter does not even have to have a bi-directional data link to the SMGW effectively mitigating any attack vector for remote compormise.

Despite these considerations we still chose to reset the application MCU inside smart meter for two reasons. One is that SMGWs are much harder to come by on the second-hand market. The other is that SMGWs are a particular feature of the German standardization landscape and in many other countries functions of an SMGW such as wireless protocol handling are integrated into the meter itself (see e.g. [77]).

In the end we settled on an Q3DA1002 three-phase 60A meter made by German manufacturer EasyMeter. This meter is typical of what would be found in an average German household and can be acquired very inexpensively as new old stock on online marketplaces.

The meter consists of a plastic enclosure with a transparent polycarbonate top part and a grey ABS bottom part that are ultrasonically welded shut. In the bottom part of the case a PCB we call the *measurement* board is potted in epoxide resin (see Figure 3.16). This PCB contains three separate energy measurement ASICs for the three phases (see Figure 3.17). It also contains a capacitive dropper power supply for the meter circuitry and external modules such as a SMGW. The measurement board through three infrared links (one per phase) communicates with a smaller unpotted PCB we call the *display* board in the top of the case. This PCB handles measurement logging and aggregation, controls a small segment LCD displaying totals and handles the externally accessible kW h impulse LED and serial IR links.

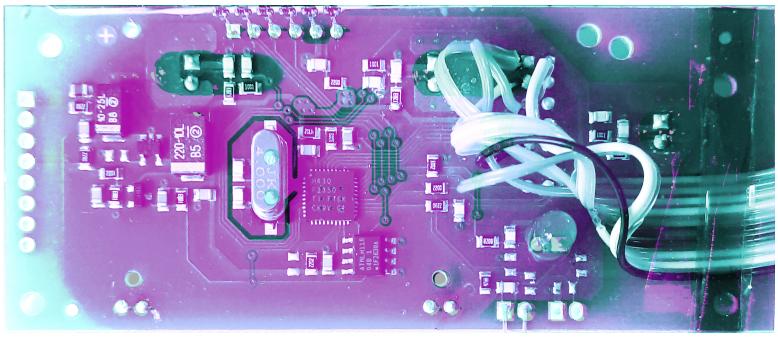
The measurement board does not contain any logging or outside communication interfaces. All of that is handled on the display board by a Texas Instruments MSP430F2350 application MCU. This is a 16-bit RISC MCU with 16 kB flash and 2 kB SRAM[8]. There is an I2C EEPROM that is used in conjunction with the microcontroller's internal 256 B data flash to keep redundant copies of energy consumption aggregates. On the side of the base board is a 14-pin header containing both a standard TI MSP430 JTAG pinout and an UART serial link for debugging. Conveniently the JTAG port was left enabled by fuse in our particular production unit.

We chose to use this MSP430 series application MCU as our reset target. Though in this particular unit compromise is impossible due to a lack of bi-directional communication links some of its sister models do contain bidirectional communication links[59] making compromise through communication interfaces at least a theoretical possibility. In other countries meters with a similar architecture to the Q3DA1002 commonly include complex protocol logic as part of the meter itself[77, 53]. As an example, the Honeywell REX2 uses a Maxim Integrated 71M6541 main application microcontroller along with a Texas Instruments CC1000 series radio transceiver and is advertised to support both over-the-air firmware upgrades and a remotely accessible "service control switch".

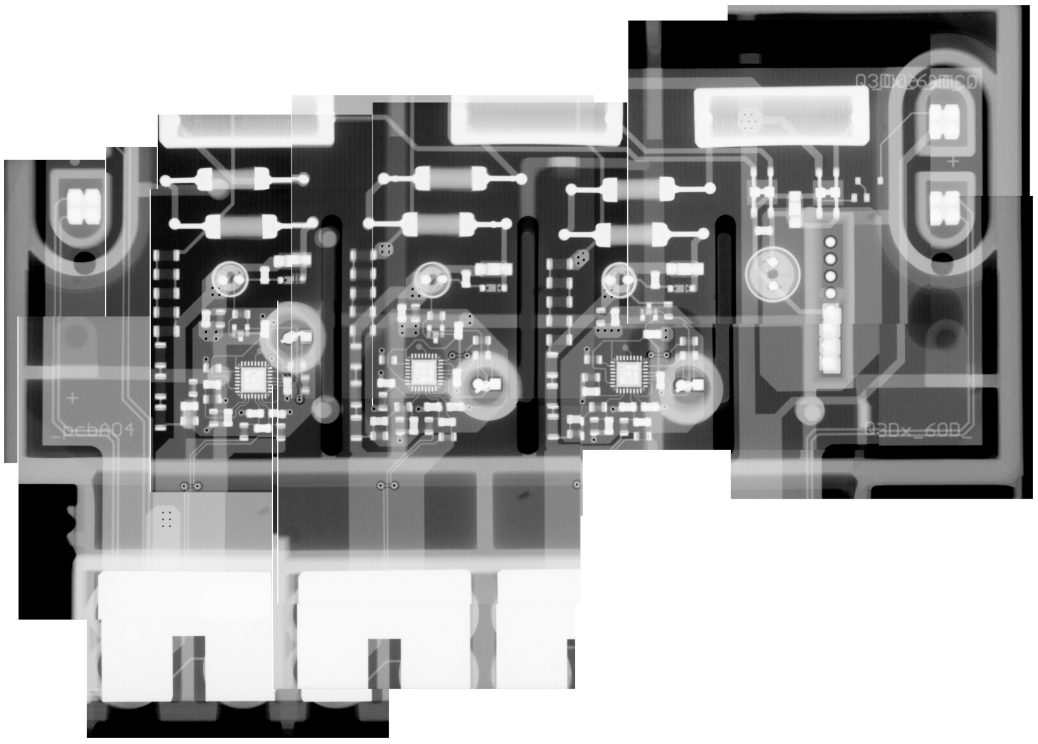
3.3.2 Firmware implementation

We based our safety reset demonstrator firmware on the grid frequency sensor firmware we developed in sec. 3.1.2. We implemented DSSS demodulation by translating the python prototype code we developed in sec. 3.2.3 to embedded C code. After validating the C translation in extensive

[8]The microcontroller might seem a bit overkill for such a simple application, but most of its 16 kB program flash is in fact used. A casual glance with Ghidra shows that a large part of program flash is expended on keeping multiple redundant copies of energy consumption aggregates including error recovery in case of data corruption and some effort has even been made to guard against data corruption using simple non-cryptographic checksums. Another large part of the MCU's firmware handles data transmission over the meter's externally accessible IR link through Smart Message Language[23].

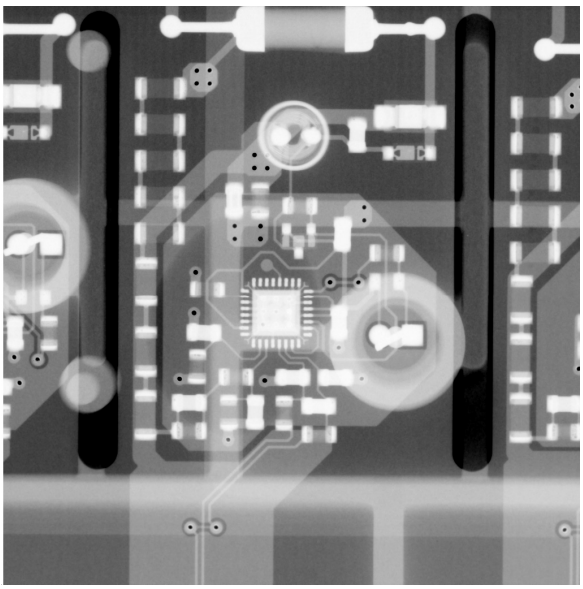


(a) Optical composite image of the display and data logging board in the top of the case. The six pins at the top are the SPI chip-on-glass segment LCD. Of the eight pads on the left six are unused and two carry the auxiliary power supply from the measurement board below. The bottom right section contains the kW h impulse LED and the angled IR communication LED. The flying wires connect to the 14-pin JTAG and serial debug header.

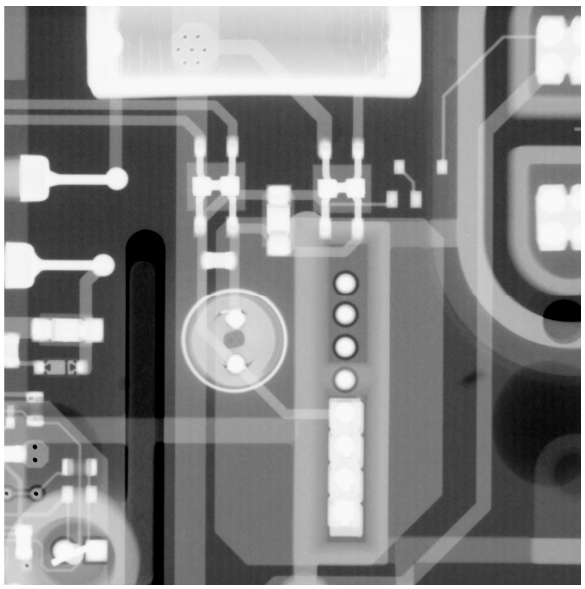


(b) Composite microfocus x-ray image of the potted measurement module in the bottom of the case. The ovals on the top left and right are power supply and data jumper connections for external modules such as SMGW interfaces. The bright parts at the bottom are the massive screw terminals with integrated current shunts. The circuitry right of the three independent measurement channels is the power supply circuit for the display board.

Figure 3.16: Composite images of the circuit boards inside the EasyMeter Q3DA1002 “smart” electricity meter used in our demonstration.



(a) Microfocus x-ray of one channel's data acquisition circuit



(b) Microfocus x-ray of the auxiliary power supply

Figure 3.17: Microfocus x-rays of major sections of the EasyMeter Q3DA1002 measurement board

simulations we integrated our code with a reed-solomon implementation and a libsodium-based implementation of the cryptographic protocol we designed in sec. 2.3.4. To reprogram the target MSP430 microcontroller we ported over the low-level bitbang JTAG driver of mspdebug[9].

For all computation-heavy high-level modules of our firmware such as the DSSS demodulator or the grid frequency estimator we wrote test fixtures that allow the same code that runs on the microcontroller to be executed on the host for testing. These test fixtures are very simple C programs that load input data from a file or the command line, run the algorithm and print results on standard output.

3.4 Grid frequency modulation emulation

To emulate a modulated grid frequency signal we superimposed a DSSS-modulated signal at the proper amplitude with synthetic grid frequency noise generated according to the measurements we took in sec. 3.1.2. In this primitive simulation we do not simulate the precise impulse response of the grid to a DSSS-modulated stimulus signal. Our results still serve to illustrate the possibility of data transmission in this manner this impulse response can be compensated for at the transmitter by selecting appropriate modulation parameters (e.g. chip rate and amplitude) and at the receiver by equalization with a matched filter.

[9] `https://github.com/dlbeer/mspdebug`

3.5 Experimental results

3.6 Lessons learned

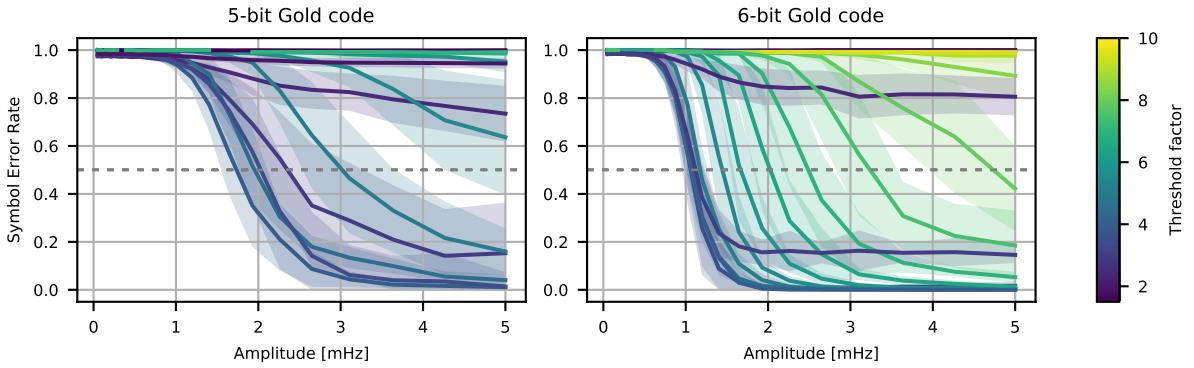
Before settling on the commercial smart meter we first tried to use an EVM430-F6779 smart meter evaluation kit made by Texas Instruments. This evaluation kit did not turn out well for two main reasons. One, it shipped with half the case missing and no cover for the terminal blocks. Because of this some work was required to maintain electrical safety. Even after mounting it in an electrically safe manner since the main MCU is not isolated from the grid and the JTAG port is also galvanically coupled the safety reset controller prototype would also have to be galvanically isolated to not pose an electrical safety risk. The second issue we ran into was that the EVM430-F6779 is based around an MSP430F6779 microcontroller. This microcontroller is a rather large part within the MSP430 series and uses a particularly new revision of the CPU core and associated JTAG peripheral that are incompatible with all MSP430 programmers we tried to use on it. mspdebug does not have support for it and porting TI's own JTAG programmer reference sources did not yield any results either. Finally we tried an USB-based programmer made by TI themselves that turned out to either have broken firmware or a hardware defect, leading to it frequently re-enumerating on the USB.

Overall our initial assumption that a development kit would certainly be easier to program than a commercial meter did not prove to be true. Contrary to our expectations the commercial meter had JTAG enabled allowing us to easily read out its stock firmware without needing to reverse-engineer vendor firmware update files or circumventing code protection measures. The fact that its firmware was only available in its compiled binary form was not much of a hindrance as it proved not to be too complex and all we wanted to know could be found out with just a few hours of digging in Ghidra.

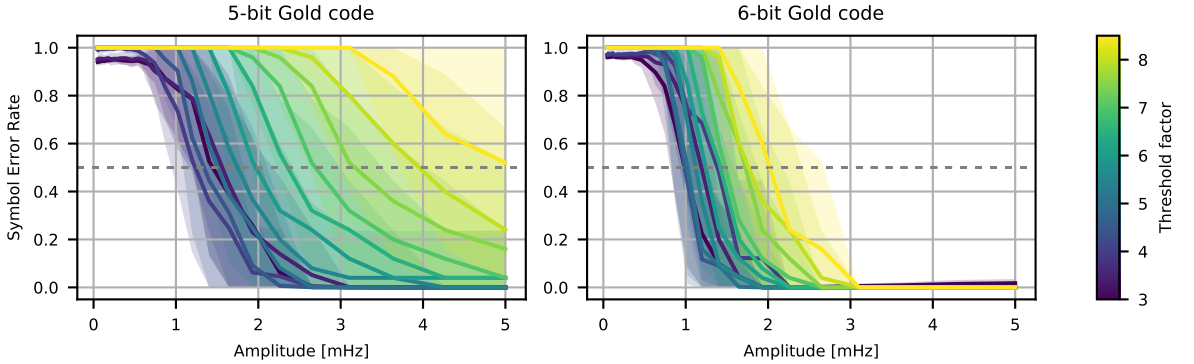
In the firmware development phase our approach of testing every module individually (e.g. DSSS demodulator, Reed-Solomon decoder, grid frequency estimation) proved to be very useful. In particular debugging benefited greatly from being able to run a couple thousand tests within seconds. In case of our DSSS demodulator this modular testing and simulation architecture allowed us to simulate many thousand runs of our implementation on test data and directly compare it to our Jupyter/Python prototype (see Figure 3.18). Since we spent more time polishing our embedded C implementation it turned out to perform much better than our initial python prototype. At the same time it shows fundamentally similar response to its parameters. One significant bug we fixed in the embedded C version is the python version's tendency towards incorrect decodings at even very large amplitudes.

In accordance with our initial estimations we did not run into any code space nor computation bottlenecks for chosing floating-point emulation instead of porting over our algorithms to fixed-point calculations. The extremely slow sampling rate of our systems makes even heavyweight processing such as FFT or our rather brute-force dynamic programming approach to DSSS demodulation possible well within performance constraints.

Compiled code size of our firmware implementation is slightly larger than we would like at around 64 kB for our firmware image including everything except the target microcontroller firmware image. See appendix C for a graph illustrating the contribution of various parts of the signal processing toolchain to this total. Overall the most heavy-weight operations by far are the SHA512 implementation from libsodium and the FFT from ARM's CMSIS signal processing library.



(a) Python prototype



(b) Embedded C implementation

Figure 3.18: Symbol error rate plots versus threshold factor for both our python prototype (above) and our firmware implementation of our demodulation algorithm. Note the slightly different threshold factor color scales. Cf. Figure 3.12.

Chapter 4

Future work

4.1 Precise grid characterization

We based our simulations on a linear relationship between generation/consumption power imbalance and grid frequency. Our literature study suggests that this is an appropriate first-order approximation. We kept modulation bandwidth in our simulations inside a 1000 mHz to 100 mHz frequency band that we reason is most likely to exibit this linear behavior in practice. At lower frequencies primary control kicks in. With the frequency delta thresholds specified for primary control systems[129] this will likely lead to significant non-linear effects. At higher frequencies grid frequency estimation at the receiver becomes more complex. Higher frequencies also come close to modes of mechanical oscillation in generators (usually at 5 Hz and above[46]).

Some limited analysis of the above concerns can be done through established dynamic grid simulation models[116, 62]. Presumably out of safety concerns these models are only available under non-disclosure agreements. Integrating even just NDA-encumbered results stemming from such a model in an open-source publication such as this one poses a logistical challenge which is why we decided to leave this topic for a separate future work. After detailed model simulation we ultimately aim to validate our results experimentally. Assuming linear grid behavior even under very small disturbances a small-scale experiment is an option. Such a small-scale experiment would require very long integration times.

Given a frequency characteristic of 30 GW Hz^{-1} a stimulus of 10 kW yields $\Delta f = 0.33\,\mu\mathrm{Hz}$. At an estimated 20 mHz of RMS noise over a bandwidth of interest this results in an SNR slightly better than −50 dB. The correlation time necessary to offset this with DSSS processing gain at a chip rate of 1 Bd would be in the order of days. With such long correlation times clock stability starts to become a problem as during correlation transmitter and receiver must maintain close phase alignment w.r.t. one chip period. A $\leq 10°$ phase difference requirement over this period of time would translate into clock stability better than 10 ppm. Though certainly not impossible to achieve this does pose an engineering challenge.

A possible way to maintain clock alignment is to use grid frequency itself as a reference. Instead of keying the DSSS modulator/demodulator on a local crystal oscillator, chip timings would be described in fractions of a mains voltage cycle. This would track grid frequency variations synchronously at both ends and would maintain phase alignment even over long periods of time at cost of a slight increase in system complexity.

4.2 Technical standardization

The description of a safety reset system provided in this work could be translated into a formalized technical standard with relatively low effort. Our system is very simple compared to e.g. a full smart meter communication standard and thus can conceivably be described in a single, concise document. The much more complicated side of standardization would be the standardization of the backend operation including key management, coördination and command authorization.

4.3 Regulatory adoption

Since the proposed system adds significant cost and development overhead at no immediate benefit to either consumer or utility company it is unlikely that it would be adopted voluntarily. Market forces limit what long-term planning utility companies can do. An advanced mitigation such as this one might be out of their reach on their own and might require regulatory intervention to be implemented. To regulatory authorities a system such as this one provides a powerful primitive to guard against attacks. Due to the low-level approach our system might allow a regulatory authority to restore meters to a safe state without the need of fine-grained control of implementation details such as application network protocols.

A regulatory authority might specify that all smart meters must use a standardized reset controller that on command resets to a minimal firmware image that disables external communication, continues basic billing functions and enables any disconnect switches. This system would enable the *reset authority* to directly preempt a large-scale attack irrespective of implementation details of the various smart meter implementations.

Cryptographic key management for the smart reset system is not much different to the management of highly privileged signing keys as they are used in many other systems already. If the safety reset system is implemented with a regulatory authority as the *reset authority* they would likely be able to find a public entity that is already managing root keys for other government systems to also manage safety reset keys. Availability and security requirements of safety reset keys do not differ significantly from those for other types of root keys.

4.4 Practical implementation

4.5 Zones of trust

In our design, we opted for a safety reset controller in form of a separate micocontroller entirely separate from whatever application microcontroller the smart meter design is already using. This design nicely separates the meter into an untrusted application (the core microcontroller) and the trusted reset controller. Since the interface between the two is simple and logically one-way, it can be validated to a high standard of security.

Despite these security benefits, the cost of such a separate hardware device might prove high in a mass-market rollout. In this case, one might attempt to integrate the reset controller into the core microcontroller in some way. Primarily, there would be two ways to accomplish this. One is a solution that physically integrates an additional microcontroller core into the main application microcontroller package either as a submodule on the same die or as a separate die in a multi-chip module (MCM) with the main application microcontroller. A full-custom solution integrating both on a single die might be a viable path for very large-scale deployments, but will

most likely be too expensive in tooling costs alone to justify its use. More likely for a medium- to large-scale deployment (millions of meters) would be a MCM integrating an off-the-shelf smart metering microcontroller die with the reset controller running on another, much smaller off-the-shelf microcontroller die. This solution might potentially save some cost compared to a solution using a discrete microcontroller for the reset controller.

The more likely approach to reducing cost overhead of the reset controller would be to employ virtualization technologies such as ARM's TrustZone in order to incorporate the reset controller firmware into the application firmware on the same chip without compromising the reset controller's security or disturbing the application firmware's operation.

TrustZone is a virtualization technology that provides a hardware-assisted privileged execution domain on at least one of the microcontrollers cores. In traditional virtualization setups a privileged hypervisor is managing several unprivileged applications sharing resources between them. Separation between applications in this setup is longitudinal between adjacent virtual machines. Two applications would both be running in unprivileged mode sharing the same cpu and the hypervisor would merely schedule them, configure hardware resource access and coördinate communication. This longitudinal virtualization simplifies application development since from the application's perspective the virtual machine looks very similar to a physical one. In addition, in general this setup reciprocally isolates two applications with neither one being able to gain control over the other.

In contrast to this, a TrustZone-like system in general does not provide several application virtual machines and longitudinal separation. Instead, it provides lateral separation between two domains: The unprivileged application firmware and a privileged hypervisor. Application firmware may communicate with the hypervisor through defined interfaces but due to TrustZone's design it need not even be aware of the hypervisor's existence. This makes a perfect fit for our reset controller. The reset controller firmware would be running in privileged mode and without exposing any communication interfaces to application firmware. The application firmware would be running in unprivileged mode without any modification. The main hurdles to the implementation to a system like this are the requirement for a microcontroller providing this type of virtualization on the one hand and the complexity of correctly employing this virtualization on the other hand. Virtualization systems such as TrustZone are still orders of magnitude more complex to correctly configure than it is to simply use separate hardware and secure the interfaces in between.

Chapter 5

Alternative use of grid frequency modulation

Chapter 6

Conclusion

Bibliography

[1] Asmaa Abdallah. *Security and Privacy in Smart Grid*. Ed. by Xuemin Shen. Cham, 2018. URL: http://dx.doi.org/10.1007/978-3-319-93677-2.

[2] Damminda Alahakoon and Xinghuo Yu. "Smart Electricity Meter Data Intelligence for Future Energy Systems: A Survey". In: *IEEE Transactions on Industrial Informatics* (2015). DOI: 10.1109/TII.2015.2414355. URL: http://ieeexplore.ieee.org.sci-hub.tw/abstract/document/7063262 (visited on).

[3] Lisa Alejandro et al. *Global Market for Smart Electricity Meters. Government Policies Driving Strong Growth*. Research rep. U.S. International Trade Commission, 2014. URL: https://www.usitc.gov/publications/332/id-037smart_meters_final.pdf (visited on 05/18/2020).

[4] Saurabh Amin et al. "Game-Theoretic Models of Electricity Theft Detection in Smart Utility Networks". In: *IEEE Control Systems Magazine* 35 (Feb. 2015). DOI: 10.1109/MCS.2014.2364711. URL: https://cloudfront.escholarship.org/dist/prd/content/qt3658w184/qt3658w184.pdf (visited on).

[5] Ross Anderson and Shailendra Fuloria. "Who controls the off switch?" In: *2010 First IEEE International Conference on Smart Grid Communications*. Gaithersburg, MD, 2010, pp. 96–101. DOI: 10.1109/SMARTGRID.2010.5622026. URL: https://www.cl.cam.ac.uk/~rja14/Papers/meters-offswitch.pdf (visited on 05/18/2020).

[6] *Application Note 200-2: Fundamentals of Quartz Oscillators*. Tech. rep. Hewlett Packard, 1997.

[7] Pol Van Aubel and Erik Poll. "Smart metering in the Netherlands: what, how and why". In: *International Journal of Electrical Power and Energy Systems* 109 (2019), pp. 719–725. ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2019.01.001.

[8] Mohammed W. Ayoub and Francis V. P. Robinson. *A comparative study between diode and thyristor based AC to DC converters for aluminium smelting process*. Tech. rep. Dubai Aluminium, 2013. DOI: https://doi.org/10.1109/IEEEGCC.2013.6705851. URL: https://purehost.bath.ac.uk/ws/files/134381670/a_comparative_study_between_diode_and_thyristor_based_AC_to_DC_converters_for_aluminium_smelting_process.pdf.

[9] Daniel Belega and Dario Petri. "Accuracy Analysis of the Multicycle Synchrophasor Estimator Provided by the Interpolated DFT Algorithm". In: *IEEE Transactions on Instrumentation and Measurement* 62 (5 2013), pp. 942–953. ISSN: 0018-9456. DOI: 10.1109/tim.2012.2236777.

[10] Matt Blaze et al. "The role of trust management in distributed systems security". In: *Secure Internet Programming*. Springer, 1999, pp. 185–210.

[11] Jozef Borkowski, Dariusz Kania, and Janusz Mroczka. "Interpolated-DFT-Based Fast and Accurate Frequency Estimation for the Control of Power". In: *IEEE Transactions on Industrial Electronics* 61 (12 2014), pp. 7026–7034. ISSN: 0278-0046. DOI: 10.1109/tie.2014.2316225.

[12] Stuart Borlase, ed. *Smart Grids: Advanced Technologies and Solutions*. Electric Power and Energy Engineering. CRC Press, 2017. ISBN: 978-1-4987-9955-3. URL: http://libgen.is/book/index.php?md5=54E49C790BF4ABE66857D6A86E60A196 (visited on).

[13] *Branchenempfehlung Strommarkt Schweiz Handbuch Smart Metering CH*. Tech. rep. Verband Schweizerischer Elektrizitätsunternehmen VSE, 2010. URL: https://web.archive.org/web/20130418034458if_/http://www.strom.ch:80/uploads/media/HBSM-CH_1018d_2010.pdf (visited on 05/12/2020).

[14] Marilyn A. Brown and Shan Zhou. "Smart-Grid Policies: An International Review. The Large-scale Renewable Energy Integration Challenge". In: *Advances in Energy Systems: The Large-scale Renewable Energy Integration Challenge*. First Ed. Wiley, 2019. DOI: 10.1002/9781119508311.

[15] Johannes Buchmann et al. "On the security of the Winternitz one-time signature scheme". In: *International Conference on Cryptology in Africa*. Springer. 2011, pp. 363–378.

[16] Bundesamt für Sicherheit in der Informationstechnik. *Marktanalyse zur Feststellung der technischen Möglichkeit zum Einbau intelligenter Messsysteme nach § 30 MsbG*. Tech. rep. Jan. 2019. URL: https://web.archive.org/web/20190919124052/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SmartMeter/Marktanalysen/Marktanalyse_nach_Para_30_MsbG.pdf?__blob=publicationFile&v=8 (visited on).

[17] Bundesamt für Sicherheit in der Informationstechnik. *Technische Richtlinie BSI TR-03109*. Bundesamt für Sicherheit in der Informationstechnik. Nov. 2015. URL: https://web.archive.org/web/20190919102010/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR03109.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=3 (visited on).

[18] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage I: CMS-Datenformat für die Inhaltsdatenverschlüsselung und -signatur*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2013. URL: https://web.archive.org/web/20190919104234/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1%5C_Anlage%5C_CMS.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2%5C_cid351?%5C_%5C_blob=publicationFile%5C&v=2 (visited on).

[19] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage II: COSEM/HTTP Webservices*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2012. URL: https://web.archive.org/web/20190919104234/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1%5C_Anlage%5C_CMS.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2%5C_cid351?%5C_%5C_blob=publicationFile%5C&v=2 (visited on).

[20] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage III: Feinspezifikation "Drahtlose LMN-Schnittstelle" Teil a: "OMS Specification Volume 2, Primary Communication"*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2013. URL: https://web.archive.org/web/20190919110054/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1_Anlage_Feinspezifikation_Drahtlose_LMN-Schnittstelle.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=2 (visited on).

[21] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage III: Feinspezifikation "Drahtlose LMN-Schnittstelle" Teil b: "OMS Technical Report Security"*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2013. URL: https://web.archive.org/web/20190919110101/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1_Anlage_Feinspezifikation_Drahtlose_LMN-Schnittstelle-Teil2.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=2 (visited on).

[22] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage IV: Feinspezifikation "Drahtgebundene LMN-Schnittstelle" Teil a: "HDLC für LMN"*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2013. URL: https://web.archive.org/web/20190919110101/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1_Anlage_Feinspezifikation_Drahtlose_LMN-Schnittstelle-Teil2.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=2 (visited on).

[23] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage IV: Feinspezifikation "Drahtgebundene LMN-Schnittstelle" Teil b: "SML Smart Message Language"*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2013. URL: https://web.archive.org/web/20190919110756/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1%5C_Anlage%5C_Feinspezifikation%5C_Drahtgebundene%5C_LMN-Schnittstelle%5C_Teilb.pdf%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2%5C_cid351?%5C_%5C_blob=publicationFile%5C&v=2 (visited on).

[24] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage VI: Betriebsprozesse*. Bundesamt für Sicherheit in der Informationstechnik. Mar. 2013. URL: https://web.archive.org/web/20190919111203/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1_Anlage_Betriebsprozesse.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=1 (visited on).

[25] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1 Anlage VII: Interoperabilitätsmodell und Geräteprofile für Smart-Meter- Gateways*. Bundesamt für Sicherheit in der Informationstechnik. Jan. 2019. URL: https://web.archive.org/web/20190919111350/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-1_Anlage_Interop-Modell-Geraeteprofile.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=2 (visited on).

[26] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-1: Anforderungen an die Interoperabilität der Kommunikationseinheit eines intelligenten Messsystems*. Bundesamt für Sicherheit in der Informationstechnik. Jan. 2019. URL: https://web.archive.org/web/20190919102217/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR03109-1.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=3 (visited on).

[27] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-2 Anhang A: Smart Meter Gateway Sicherheitsmodul Use Cases*. Bundesamt für Sicherheit in der Informationstechnik. Dec. 2014. URL: https://web.archive.org/web/20190919111540/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-2-Sicherheitsmodul_Use_Cases.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=2 (visited on).

[28] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-2 Anhang B: Smart Meter Mini-HSM Anforderungen an die Funktionalität und Interoperabilität des Sicherheitsmoduls*. Bundesamt für Sicherheit in der Informationstechnik. June 2017. URL: https://web.archive.org/web/20190919111832/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-2_Anhang_B_Smart_Meter_Mini_HSM.pdf;%20jsessionid=BD197BE4CB44C76EE794564
2_cid351?__blob=publicationFile&v=3 (visited on).

[29] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-2: Smart Meter Gateway - Anforderungen an die Funktionalität und Interoperabilität des Sicherheitsmoduls*. Bundesamt für Sicherheit in der Informationstechnik. Dec. 2014. URL: https://web.archive.org/web/20190919102644/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-2-Anforderungen_an_die_Funktionalitaet.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B87
2_cid351?__blob=publicationFile&v=3 (visited on).

[30] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-3: Kryptographische Vorgaben für die Infrastruktur von intelligenten Messsystemen*. Bundesamt für Sicherheit in der Informationstechnik. Apr. 2014. URL: https://web.archive.org/web/20190919102648/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-3_Kryptographische_Vorgaben.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=1 (visited on).

[31] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-4: Public Key Infrastruktur für Smart Meter Gateways*. Bundesamt für Sicherheit in der Informationstechnik. Aug. 2017. URL: https://web.archive.org/web/20190919102649/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-4_PKI.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=3 (visited on).

[32] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-6: Smart Meter Gateway Administration*. Bundesamt für Sicherheit in der Informationstechnik. Nov. 2015. URL: https://web.archive.org/web/20190919102651/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/

TR03109/TR-03109-6-Smart_Meter_Gateway_Administration.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=4 (visited on).

[33] Bundesamt für Sicherheit in der Informationstechnik. *TR-03109-TS-1: Testkonzept zu BSI TR-03109-1*. Bundesamt für Sicherheit in der Informationstechnik. Jan. 2015. URL: https://web.archive.org/web/20190919112310/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/TR-03109-TS-1_Testkonzept.pdf;%20jsessionid=BD197BE4CB44C76EE7945640B8703844.2_cid351?__blob=publicationFile&v=1 (visited on).

[34] Bundesamt für Sicherheit in der Informationstechnik. *TR-03116-3: Intelligente Messsysteme*. Bundesamt für Sicherheit in der Informationstechnik. Jan. 2019. URL: https://web.archive.org/web/20190919112052/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-3.pdf;%20jsessionid=CB56FC0D3137C5624CA697AB9E57671F.1_cid360?__blob=publicationFile&v=9 (visited on).

[35] Bundesamt für Sicherheit in der Informationstechnik. *TR-Prüfstellen: Anforderungen an Antragsteller zur Anerkennung als Prüfstelle im Bereich Technischer Richtlinien*. Bundesamt für Sicherheit in der Informationstechnik. Jan. 2019. URL: https://web.archive.org/web/20190919112552/https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/TR-Pruefstellen.pdf;%20jsessionid=A6B4CB8AD2C038741C656276CE8742_cid369?__blob=publicationFile&v=10 (visited on).

[36] Bundesamt für Sicherheit in der Informationstechnik and Bundesministerium für Wirtschaft und Energie. *Standardisierungsstrategie zur sektorübergreifenden Digitalisierung nach dem Gesetz zur Digitalisierung der Energiewende*. Jan. 2019. URL: https://web.archive.org/web/20190919100713/https://www.bmwi.de/Redaktion/DE/Downloads/S-T/standardisierungsstrategie.pdf?__blob=publicationFile&v=4 (visited on 09/19/2019).

[37] Bundesministerium für Wirtschaft und Energie. *Häufig gestellte Fragen rund um Smart Meter*. Apr. 14, 2020. URL: https://www.bmwi.de/Redaktion/DE/FAQ/Smart-Meter/faq-smart-meter.html (visited on 05/20/2020).

[38] Bundesministerium für Wirtschaft und Energie and Ernst and Young. *Kosten-Nutzen-Analyse für einen flächendeckenden Einsatz intelligenter Zähler*. Tech. rep. 2013. URL: https://www.bmwi.de/Redaktion/DE/Publikationen/Studien/kosten-nutzen-analyse-fuer-flaechendeckenden-einsatz-intelligenterzaehler.pdf?__blob=publicationFile&v=5 (visited on 05/12/2020).

[39] Bundesnetzagentur. *Smart Meter*. 2019. URL: https://web.archive.org/web/20190919100204/https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Verbraucher/NetzanschlussUndMessung/SmartMetering/SmartMeter_node.html (visited on 09/19/2019).

[40] Stuart Cheshire and Mary Baker. "Consistent overhead Byte stuffing". In: *IEEE/ACM Trans. Netw.* 7.2 (1999), pp. 159–172. DOI: 10.1109/90.769765.

[41] The European Commission, ed. *The Energy Efficiency Directive*. 2012. URL: https://ec.europa.eu/energy/topics/energy-efficiency/targets-directive-and-rules/energy-efficiency-directive_en (visited on 05/18/2020).

[42] Liviu Constantinescu-Simon, ed. *Handbuch Elektrische Energietechnik*. 1997. DOI: 10.1007/978-3-322-85061-4.

[43] Enrico Costanza et al. "Doing the Laundry with Agents: a Field Trial of a Future SmartEnergy System in the Home. a field trial of a future smart energy system in the home". In: *CHI 2014, One of a CHInd*. 2014. DOI: 10.1145/2556288.2557167.

[44] Adrian Cramer. *Bedienhandbuch "Smart Meter"*. NETZE Bad Langensalza GmbH. 2015. URL: https://www.nbl-badlangensalza.de/fileadmin/Netze/Dokumente/Benutzerhandbuch_NBL-Smart_Meter_V.1.0_.pdf.

[45] Valentin Crastan. *Elektrische Energieversorgung 1*. 2015. DOI: 10.1007/978-3-662-45985-0.

[46] Valentin Crastan. *Elektrische Energieversorgung 3*. 2012. ISBN: 978-3-642-20099-1. DOI: 10.1007/978-3-642-20100-4.

[47] Colette Cuijpers and Bert-Jaap Koops. "Smart metering and privacy in Europe: lessons from the Dutch case". In: *European data protection. Coming of age* (2012), pp. 269–293. DOI: https://doi.org/10.1007/978-94-007-5170-5_12.

[48] R. Czechowski and A. M. Kosek. "The most frequent energy theft techniques and hazards in present power energy consumption". In: *2016 Joint Workshop on Cyber-Physical Security and Resilience in Smart Grids (CPSR-SG)*. IEEE. Apr. 2016, pp. 1–7. DOI: 10.1109/CPSRSG.2016.7684098. URL: https://project-sparks.eu/wp-content/uploads/2016/04/czechowski-cpsr-sg-paper-four.pdf.

[49] Mathias Dalheimer. *Smartin Meter-Einführung Deutschland*. URL: https://entropia.de/images/2/2c/GPN14-SmartMeterEinf%C3%BChrung.pdf.

[50] Antonio Del Giudice et al. "Power quality in smart distribution grids". In: *2015 IEEE International Workshop on Measurements & Networking (M&N)*. IEEE. 2015, pp. 1–6.

[51] Asja Derviškadić, Paolo Romano, and Mario Paolone. "Iterative-interpolated DFT for synchrophasor estimation: A single algorithm for P-and M-class compliant PMUs". In: *IEEE Transactions on Instrumentation and Measurement* 67.3 (2017), pp. 547–558.

[52] Statistisches Bundesamt DeStatis, ed. *Erzeugung - Bilanz - Monatsbericht über die Elektrizitätsversorgung*. Mar. 6, 2020. URL: https://www.destatis.de/DE/Themen/Branchen-Unternehmen/Energie/Erzeugung/Tabellen/bilanz-elektrizitaetsversorgung.html (visited on 05/07/2020).

[53] Miro Djuric. *Elster REX2 Smart Meter Teardown*. iFixit. 2011. URL: https://www.ifixit.com/News/14306/elster-rex2-smart-meter-teardown (visited on 05/06/2020).

[54] Chris Dods, Nigel P Smart, and Martijn Stam. "Hash based digital signature schemes". In: *IMA International Conference on Cryptography and Coding*. Springer. 2005, pp. 96–115.

[55] Roman Düssel. "Paradigm Shift in the Indication of a Stable Cell during Power Modulation". In: *Proceedings of the 36th International ICSOBA Conference*. 2018.

[56] *Dutch Smart Meter Requirements P1 Companion Standard*. Tech. rep. Version 5.0.2. Netbeheer Nederland WG DSMR, 2016. URL: https://smarty.creos.net/wp-content/uploads/DutchSmartMeterRequirements.pdf (visited on 05/18/2020).

[57] *Dutch Smart Meter Requirements P3 Companion Standard*. Tech. rep. Version 4.0.7. Netbeheer Nederland WG DSMR, 2014. URL: https://www.netbeheernederland.nl/_upload/Files/Slimme_meter_15_1f3c5c9b2c.pdf (visited on 05/18/2020).

[58] Dacfey Dzung, Inigo Berganza, and Alberto Sendin. "Evolution of powerline communications for smart distribution: From Ripple Control to OFDM". In: *2011 IEEE International Symposium on Power Line Communications and Its Applications* (2011). DOI: 10.1109/ISPLC.2011.5764444. URL: https://www.researchgate.net/profile/Inigo_Berganza/publication/224236306_Evolution_of_powerline_communications_for_smart_distribution_From_ripple_control_to_OFDM/links/5c658800299bf1d14cc74cbd/Evolution-of-powerline-communications-for-smart-distribution-From-ripple-control-to-OFDM.pdf.

[59] EasyMeter GmbH. *Datenblatt Moderne Messeinrichtung Q3A Drehstromzähler*. 2020.

[60] Christian Egenhofer et al. *Composition and Drivers of Energy Prices and Costs: Case Studies in SelectedEnergy Intensive Industries – 2018*. Tech. rep. European Commission, Directorate-General for Internal Market, Industry, Entrepreneurship and SMEs, 2018. DOI: 10.2873/937326. URL: https://op.europa.eu/en/publication-detail/-/publication/424dac0a-ec77-11e8-b690-01aa75ed71a1/language-en.

[61] David Eisma and Pretesh Patel. "Challenges in Power Modulation". In: *Essential Readings in Light Metals, Volume 2, Aluminum Reduction Technology*. Ed. by Geoff Bearne, Marc Dupuis, and Gary Tarcy. 2016, pp. 683–688.

[62] ENTSO-E. *ENTSO-E Initial Dynamic Model of Continental Europe*. 2019. URL: https://www.entsoe.eu/publications/system-operations-reports/#entso-e-initial-dynamic-model-of-continental-europe (visited on 05/14/2020).

[63] ENTSO-E System Protection Dynamics and WG. *Oscillation Event 03.12.2017*. Tech. rep. Mar. 2018. URL: https://docstore.entsoe.eu/Documents/SOC%20documents/Regional_Groups_Continental_Europe/OSCILLATION_REPORT_SPD.pdf.

[64] ENTSO-E Working Group Incident Classification Scale Under System Operations Committee. *INCIDENTS CLASSIFICATIONSCALEMETHODOLOGY*. Tech. rep. ENTSO-E, 2014.

[65] Michael J. Fell et al. "Public acceptability of domestic demand-side response in Great Britain: The role of automation and direct load control". In: *Energy Research and Social Science* 9 (2015), pp. 72–84. ISSN: 2214-6296. DOI: 10.1016/j.erss.2015.08.023.

[66] *Introducing GAMfIS: A Generic Attacker Model for Information Security*. IEEE, Nov. 2017. URL: https://doi.org/10.23919/SOFTCOM.2017.8115550.

[67] Jochen Fritz and Alexander Hovi. *Transkommando-System*. 2020. URL: http://www.rundsteuerung.de/entwicklung/transkommando.html.

[68] M Gasior and JL Gonzalez. *Improving FFT frequency measurement resolution by parabolic and gaussian interpolation*. Tech. rep. CERN-AB-Note-2004-021, 2004.

[69] Marek Gasior. "Improving frequency resolution of discrete spectra: algorithms of three-node interpolation". 2006. URL: https://cds.cern.ch/record/1346070.

[70] Daphne Geelen et al. "The use of apps to promote energy saving: a study of smart-meter–related feedback in the Netherlands". In: *Energy Efficiency* (12 2019). DOI: https://doi.org/10.1007/s12053-019-09777-z.

[71] Alois M. J. Goiser. *Handbuch der Spread-Spectrum Technik*. Springer, 1998. ISBN: 3-211-83080-4.

[72] *Low Frequency Oscillations in the Interconnected System of Continental Europe*. IEEE, Aug. 2010. DOI: 10.1109/PES.2010.5589932{\textperiodcentered}.

[73] Ulrich Greveler et al. "Multimedia Content Identification Through SmartMeter Power Usage Profiles". In: *Computers, Privacy and Data Protection* (2012).

[74] Vehbi C. Güngör et al. "Smart Grid Technologies: Communication Technologies and Standards". In: *IEEE Transactions on Industrial Informatics* 7.4 (Nov. 2011), pp. 529–539. URL: https://www.researchgate.net/profile/Salih_Ergut/publication/224257498_Smart_Grid_Technologies_Communication_Technologies_and_Standards/links/56ccb4e508ae85c8233bc062/Smart-Grid-Technologies-Communication-Technologies-and-Standards.pdf.

[75] *Gutachten Digitalisierung der Energiewende*. Tech. rep. Bundesministeriums für Wirtschaft und Energie, 2019. URL: https://www.bmwi.de/Redaktion/DE/Publikationen/Studien/digitalisierung-der-energiewende-thema-1.pdf?__blob=publicationFile&v=4.

[76] Hager Group. "Hager Smart Meter EHZ363 Betriebsanleitung". 2017. URL: https://bnnetze.de/downloads/kunden/netzkunden/messstellenbetrieb-und-messung/funktionalitaet/hager-ehz363-betriebsanleitung.pdf (visited on 05/11/2020).

[77] Honeywell Smart Energy. *Datasheet Honeywell REX2 smart meter*. 2017. URL: https://www.elstersolutions.com/assets/products/products_elster_files/SEADSNAEN001017REX2.pdf.

[78] Mitsuhide Ishima, Kiyoyuki Terai, and Yoshihiro Ogita. *Construction and Operation of Communication System for Smart Meter System of TEPCO Power Grid, Inc.* Tech. rep. Toshiba Energy Systems and Solutions, 2018, pp. 46–50. URL: https://www.toshiba.co.jp/tech/review/2018/04/73_04pdf/f02.pdf (visited on 05/18/2020).

[79] Itron Inc. "Benutzerhandbuch Smart Meter EM 214". 2012. URL: https://www.ewh.de/fileadmin/user_upload/Stromnetz/Zaehlerstaende/Produktbeschreibung_ITRON_EM214.pdf (visited on 05/11/2020).

[80] Yasin Kabalci. "A survey on smart metering and smart grid communication". In: *Renewable and Sustainable Energy Reviews* 57 (2016), pp. 302–318. DOI: 10.1016/j.rser.2015.12.114. URL: https://www.researchgate.net/profile/Yasin_Kabalci/publication/289504234_A_survey_on_smart_metering_and_smart_grid_communication/links/5a6105aaaca272a1581745c1/A-survey-on-smart-metering-and-smart-grid-communication.pdf.

[81] Kamstrup A/S. *STS prepayment meter*. URL: https://www.kamstrup.com/en-en/electricity-solutions/smart-electricity-meters/sts-prepayment-meter (visited on 05/11/2020).

[82] Abraham Kaplan. "The Law of the Instrument". In: *The Conduct of Inquiry: Methodology for Behavioral Science*. San Francisco: Chandler Publishing Co., 1964, p. 28. ISBN: 9781412836296. URL: https://books.google.com/books?id=OYe6fsXSP3IC&pg=PA28.

[83] Jinsub Kim and Lang Tong. "On Topology Attack of a Smart Grid: Undetectable Attacks and Countermeasures". In: *IEEE Journal on Selected Areas in Communications* 31.7 (July 2013). DOI: 10.1109/JSAC.2013.130712.

[84] Thomas Kluyver et al. "Jupyter Notebooks - a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing, Göttingen, Germany, June 7-9, 2016*. Ed. by Fernando Loizides and Birgit Schmidt. IOS Press, 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

[85] Oliver Kosut et al. "Malicious Data Attacks on the Smart Grid". In: *IEEE Transactions on Smart Grid* 2.4 (Nov. 2011), pp. 645–658.

[86] Andrew E. Kramer. *How the Kremlin Recruited an Army of Specialists for Cyberwar*. 2016.

[87] Prabha Kundur. *Power system stability and control*. eng. The EPRI power system engineering series. New York, NY u.a.: McGraw-Hill, 1994. ISBN: 007035958X.

[88] Leslie Lamport. *Constructing digital signatures from a one-way function*. Tech. rep. Technical Report CSL-98, SRI International, 1979.

[89] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (July 1982), pp. 382–401. URL: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf;%20https://doi.org/10.1145%2F357172.357176.

[90] Robert M. Lee, Michael J. Assante, and Tim Conway. "Analysis of the cyber attack on the Ukrainian power grid". In: *Electricity Information Sharing and Analysis Center (E-ISAC)* (2016).

[91] Javier Leiva, Alfonso Palacios, and José A. Aguado. "Smart metering trends, implications and necessities: A policy review". In: *Renewable and Sustainable Energy Reviews* 55 (2016), pp. 227–233. URL: http://kchbi.chtf.stuba.sk/upload_new/file/Miro/Proc%20problemy%20odovzdane%20zadania/Cyprichov%C3%A1/SmartMetering.pdf;%20http://dx.doi.org/10.1016/j.rser.2015.11.002 (visited on).

[92] Nancy G. Leveson and Clark S. Turner. "An Investigation of the Therac-25 Accidents". In: *IEEE Computer* 26.7 (July 1993), pp. 18–41. URL: https://doi.org/10.1109/MC.1993.274940;%20https://web.archive.org/web/20041128024227/http://www.cs.umd.edu/class/spring2003/cmsc838p/Misc/therac.pdf.

[93] Jaime Lloret et al. *An Integrated IoT Architecture for Smart Metering*. IEEE, 2016. (Visited on).

[94] G. Lopez et al. "Paving the road toward Smart Grids through large-scale advanced metering infrastructures". In: *Electric Power Systems Research* (2014). DOI: 10.1016/j.epsr.2014.05.006. URL: http://www.sciencedirect.com.sci-hub.tw/science/article/abs/pii/S0378779614001862 (visited on).

[95] Deborah Lupton. "The diverse domains of quantified selves: self-tracking modes and dataveillance". In: *Economy and Society* 45 (2016), pp. 101–122. ISSN: 0308-5147. DOI: 10.1080/03085147.2016.1143726.

[96] Peter Mahlknecht. “Diplomarbeit Sicherheitsmodul für ein Smart Metering Gateway”. Technische Universität Wien, 2014. URL: https://publik.tuwien.ac.at/files/PubDat_233035.pdf (visited on 05/18/2020).

[97] Anzar Mahmood, Nadeem Javaid, and Sohail Razzaq. “A review of wireless communications for smart grid”. In: *Renewable and Sustainable Energy Reviews* 41 (2015), pp. 248–260. DOI: 10.1016/j.rser.2014.08.036. URL: http://www.sciencedirect.com.sci-hub.tw/science/article/abs/pii/S1364032114007126 (visited on).

[98] Heise Medien. *checkm8: Boot-Exploit soll neuere iPhones knacken.* URL: https://www.heise.de/mac-and-i/meldung/checkm8-Boot-Exploit-soll-neuere-iPhones-knacken-4542075.html.

[99] Ralph C Merkle. “A certified digital signature”. In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 218–238.

[100] Hermann Merz, Thomas Hansemann, and Christof Hübner. *Building automation. Communication systems with EIB/KNX, LON, and BACnet*. Springer series on signals and communication technology. Berlin [u.a.]: Springer, 2009. X, 282. ISBN: 9783540888284.

[101] Yilin Mo et al. “Cyber-Physical Security of a Smart Grid Infrastructure”. In: *Proceedings of the IEEE* 100.1 (Jan. 2012), pp. 195–209. URL: http://ieeexplore.ieee.org.sci-hub.tw/abstract/document/6016202;%20https://ieeexplore.ieee.org/abstract/document/6016202;%20https://www.researchgate.net/profile/Yilin_Mo/publication/224257991_Cyber-Physical_Security_of_a_Smart_Grid_Infrastructure/links/004635395d2f66a584000000.pdf (visited on).

[102] Andrés Molina-Markham et al. “Private Memoirs of a Smart Meter”. In: *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building* (2010). Title from The ACM Digital Library. URL: https://dl.acm.org/doi/pdf/10.1145/1878431.1878446.

[103] Claudio Narduzzi et al. “Fast-TFM—Multifrequency phasor measurement for distribution networks”. In: *IEEE Transactions on Instrumentation and Measurement* 67.8 (2018), pp. 1825–1835.

[104] Dieter Nelles and Christian Tuttas. *Elektrische Energietechnik*. 1998. ISBN: 978-3-663-09902-4. DOI: 10.1007/978-3-663-09902-4.

[105] Harald A. Øye. *Power Failure, Temporary Pot Shut-Down, Restart and Repair*. Tech. rep. 27th International Aluminium Conference Metal Bulletin Events, 2012.

[106] Dillon Pariente and Emmanuel Ledinot. *Formal verification of industrial C code using Frama-C: a case study*. Tech. rep. 2010, pp. 205–219.

[107] McDaniel Patrick and McLaughlin Stephen. “Security and Privacy Challenges in the Smart Grid”. In: *Secure Systems* (May 2009).

[108] Trevor Perrin. “The Noise protocol framework, 2015”. In: *URL http://noiseprotocol.org/noise. pdf* (2016).

[109] James Pierce and Eric Paulos. “Beyond Energy Monitors: Interaction, Energy, and Emerging Energy Systems. interaction, energy, and emerging energy systems”. In: *CHI 2012*. 2012. DOI: 10.1145/2207676.2207771.

[110] Tom A. Rodden et al. “At Home with Agents: Exploring Attitudes Towards Future Smart Energy Infrastructures”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. 2013. DOI: 10.1145/2470654.

[111] Graham Rogers. “Power System Oscillations”. In: Kluwer, 2000.

[112] Peter H. Salus. *A Quarter Century of UNIX*. Addison-Wesley, 1994. ISBN: 0201547775, 9780201547771.

[113] Takuro Sato et al. *Smart Grid Standards. Specifications, Requirements and Technologies*. Wiley, 2015.

[114] Benjamin Schäfer et al. “Decentral Smart Grid Control”. In: *New Journal of Physics* 17 (Jan. 2015). DOI: doi:10.1088/1367-2630/17/1/015002.

[115] Bruce Schneier. *Secrecy, Security, and Obscurity*. May 2002. URL: https://www.schneier.com/crypto-gram/archives/2002/0515.html.

[116] Anatoli Semerow et al. *Dynamic Study Model for the interconnected power system of Continental Europe in different simulation tools*. Tech. rep. University of Erlangen-Nuremberg and ENTSO-E, 2015. DOI: 10.1109/ptc.2015.7232578.

[117] Konark Sharma and Lalit Mohan Saini. “Performance analysis of smart metering for smart grid: An overview”. In: *Renewable and Sustainable Energy Reviews* 49 (2015), pp. 720–735. DOI: 10.1016/j.rser.2015.04.170. URL: http://www.sciencedirect.com.sci-hub.tw/science/article/abs/pii/S1364032115004402 (visited on).

[118] *Single Market Progress Report: Country Profiles – Italy*. Research rep. European Commission, 2014. URL: https://ec.europa.eu/energy/sites/ener/files/documents/2014_countryreports_italy.pdf (visited on 05/18/2020).

[119] *Smart Meter Rollout Cost-Benefit Analysis Part I*. Tech. rep. UK Department for Business, Energy and Industrial Strategy, 2016. URL: https://ec.europa.eu/growth/tools-databases/tris/cs/index.cfm/search/?trisaction=search.detail&year=2017&num=350&iLang=EN (visited on 05/18/2020).

[120] *Smart Metering Implementation Programme Progress Report for 2018*. Tech. rep. UK Department for Business, Energy and Industrial Strategy, 2018. URL: https://www.gov.uk/government/publications/smart-metering-implementation-programme-progress-report-2018 (visited on 05/18/2020).

[121] *Smart Metering Implementation ProgrammeSmart Metering Equipment Technical Specifications*. Tech. rep. Version 1.58. UK Department of Energy and Climate Change, 2014. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/381535/SMIP_E2E_SMETS2.pdf (visited on 05/18/2020).

[122] ST Microelectronics. *STSAFE-J100-BS Data brief. Security module of a smart meter gateway as defined by the BSI*. ST Microelectronics, June 2018. URL: https://www.st.com/resource/en/data_brief/stsafe-j100-bs.pdf (visited on).

[123] Armin Steinbach and Michael Weise, eds. *MsbG : Kommentar zum Messstellenbetriebsgesetz*. De Gruyter, 2018. ISBN: 9783110555882.

[124] *Stromgrundversorgungsverordnung StromGVV § 19 Unterbrechung der Versorgung*. URL: http://www.gesetze-im-internet.de/stromgvv/__19.html (visited on 05/18/2020).

[125] Michael Stuber. "Standards, Security, and Smart Meters". In: *Smart Grid Handbook*. 2016, pp. 1–14. DOI: 10.1002/9781118755471.sgd036.

[126] Chih-Che Sun, Adam Hahn, and Chen-Ching Liu. "Cyber security of a power grid: State-of-the-art". In: *International Journal of Electrical Power & Energy Systems* 99 (2018), pp. 45–56.

[127] The CEN/CENELEC/ETSI Joint Working Group Standards Smart on for Grids. *Final report of the CEN/CENELEC/ETSI Joint Working Group on Standards for Smart Grids*. Tech. rep. May 2011.

[128] Frédéric Tounquet and Clément Alaton. *Benchmarking smart meteringdeployment in the EU-28*. Research rep. European Commission, Directorate-General for Energy, Directorate B - Internal Energy Market, 2019.

[129] UCTE/ENTSO-E. *Operation Handbook*. Tech. rep. 2004. Chap. Appendix 1: Load-Frequency Control and Performance.

[130] UCTE/ENTSO-E. *Operation Handbook*. Tech. rep. 2009. Chap. Policy 1: Load-Frequency Control-Final Version (approved by SC on 19 March 2009).

[131] Noelia Uribe-Pérez et al. "State of the Art and Trends Review of Smart Metering in Electricity Grids". In: *Applied Sciences* 6.3 (Feb. 2016), p. 68. DOI: 10.3390/app6030068.

[132] Andoni Urtasun et al. "Energy management strategy for a battery-diesel stand-alone system with distributed PV generation based on grid frequency modulation". In: *Renewable Energy* 66 (Jan. 2014), pp. 325–336. URL: http://dx.doi.org/10.1016/j.renene.2013.12.020.

[133] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[134] Silicon Labs Vivek Mohan. *An Introduction to Wireless M-Bus*. 2015. URL: http://pages.silabs.com/rs/634-SLU-379/images/introduction-to-wireless-mbus.pdf.

[135] Loren Weith. "DLMS / COSEM Protocol Security Evaluation". MA thesis. Department of Mathematics and Computer Science, Eindhoven University of Technology, 2014. URL: https://pure.tue.nl/ws/files/46962657/773263-1.pdf (visited on 05/20/2020).

[136] Paul S. Wright. *Library of ROCOF Test Waveforms – Pseudo Code, V1.0, May 2019*. Tech. rep. UK National Physical Laboratory, 2019. DOI: 10.5281/zenodo.3559798.

[137] Yongdong Wu et al. "Resonance Attacks on Load Frequency Control of Smart Grids". In: *IEEE Transactions on Smart Grid* 9.5 (Sept. 2018), pp. 4490–4502. DOI: 10.1109/TSG.2017.2661307.

[138] Ye Yan et al. "A Survey on Smart Grid Communication Infrastructures: Motivations, Requirements and Challenges". In: *IEEE Communications Surveys & Tutorials* (2012). DOI: 10.1109/SURV.2012.021312.00034. URL: http://d-scholarship.pitt.edu/12508/1/Smart_Grid_Infrastructure_Final.pdf.

[139] Jixuan Zheng, David Wenzhong Gao, and Li Lin. "Smart meters in smart grid: An overview". In: *2013 IEEE Green Technologies Conference (GreenTech)*. IEEE. 2013, pp. 57–64.

[140] Bin Zhou et al. “Smart home energy management systems: Concept, configurations, and scheduling strategies”. In: *Renewable and Sustainable Energy Reviews* 61 (2016), pp. 30–40. URL: http://www.sciencedirect.com.sci-hub.tw/science/article/abs/pii/S1364032116002823.

[141] Shoshana Zuboff. *The Age of Surveillance Capitalism*. 2019.

Appendix A

Transcripts of Jupyter notebooks used in this thesis

A.1 Grid frequency estimation

1 Setup

1.1 Import required packages

```
In [1]: import math
        import sqlite3
        import struct
        import datetime
        import json

        import scipy.fftpack
        from scipy import signal as sig
        import matplotlib
        from matplotlib import pyplot as plt
        from matplotlib import patches
        from matplotlib import dates
        import numpy as np
        from scipy import signal, optimize, interpolate
        from tqdm.notebook import tnrange, tqdm
        from IPython.display import set_matplotlib_formats
```

```
In [2]: %matplotlib inline
        set_matplotlib_formats('png', 'pdf')
        font = {'family' : 'normal',
                'weight' : 'normal',
                'size'   : 10}
        matplotlib.rc('font', **font)
```

1.2 Load data series information from sqlite capture file

One capture file may contain multiple runs/data series. Display a list of runs and their start/end time and sample count, then select the newest one in `last_run` variable.

```
In [3]: db = sqlite3.connect('data/waveform-raspi-ocxo-2day.sqlite3')
```

```
In [4]: for run_id, start, end, count in db.execute('SELECT run_id, MIN(rx_ts), MAX(rx_ts), COU
            foo = lambda x: datetime.datetime.fromtimestamp(x/1000)
            start, end = foo(start), foo(end)
            time_window = f'{start:%Y-%m-%d %H:%M:%S} - {end:%Y-%m-%d %H:%M:%S}'
            print(f'Run {run_id:03d}: {time_window} ({str(end-start)[:-3]:>13}, {count*32:>9d}s
        last_run, n_records = run_id, count
```

```
Run 000: 2020-04-01 14:00:25 - 2020-04-01 15:09:31 (  1:09:05.846,   4197664sp)
Run 001: 2020-04-02 11:56:41 - 2020-04-02 11:57:59 (  0:01:18.544,     79552sp)
```

```
Run 002: 2020-04-02 12:03:51 - 2020-04-03 14:12:18 (1 day, 2:08:27.618,  95262592sp)
Run 003: 2020-04-03 14:12:48 - 2020-04-06 10:33:25 (2 days, 20:20:36.644, 249113600sp)
```

1.3 Setup analog parameters

Setup parameters of analog capture hardware here. This is used to scale samples from ADC counts to analog voltages. Also setup sampling rate here. Nominal sampling rate is 1 ksps.

In [5]:
```
sampling_rate = 1000.0 * 48.6 / 48

par = lambda *rs: 1/sum(1/r for r in rs) # resistor parallel calculation

# Note: These are for the first prototype only!
vmeas_source_impedance = 330e3
vmeas_source_scale = 0.5

vcc = 15.0
vmeas_div_high = 27e3
vmeas_div_low = par(4.7e3, 10e3)
vmeas_div_voltage = vcc * vmeas_div_low / (vmeas_div_high + vmeas_div_low)
vmeas_div_impedance = par(vmeas_div_high, vmeas_div_low)

#vmeas_overall_factor = vmeas_div_impedance / (vmeas_source_impedance + vmeas_div_impe
v0 = 1.5746
v100 = 2.004
vn100 = 1.1452

adc_vcc = 3.3 # V
adc_fullscale = 4095

adc_val_to_voltage_factor = 1/adc_fullscale * adc_vcc

adc_count_to_vmeas = lambda x: (x*adc_val_to_voltage_factor - v0) / (v100-v0) * 100
```

1.4 Load run data from sqlite3 capture file

Load measurement data for the selected run and assemble a numpy array containing one continuous trace.

In [6]:
```
limit = n_records
record_size = 32
skip_dropped_sections = False

data = np.zeros(limit*record_size)
data[:] = np.nan

last_seq = None
```

```
        write_index = 0
        for i, (seq, chunk) in tqdm(enumerate(db.execute(
                'SELECT seq, data FROM measurements WHERE run_id = ? ORDER BY rx_ts LIMIT ? OF
                (last_run, limit, n_records-limit))), total=n_records):

            if last_seq is None or seq == (last_seq + 1)%0x10000:
                last_seq = seq
                idx = write_index if skip_dropped_sections else i
                data[idx*record_size:(idx+1)*record_size] = np.frombuffer(chunk, dtype='<H')
                write_index += 1

            elif seq > last_seq:
                last_seq = seq
                # nans = np.empty((record_size,))
                # nans[:] = np.nan
                # data = np.append(data, nans) FIXME

        data = (data * adc_val_to_voltage_factor - v0) / (v100-v0) * 100

        # https://stackoverflow.com/questions/6518811/interpolate-nan-values-in-a-numpy-array
        nan_helper = lambda y: (np.isnan(y), lambda z: z.nonzero()[0])

        # data rarely may contain NaNs where the capture script failed to read and acknowledge
        # For RMS calculation and overall FFT fill these NaNs with interpolated values from th
        data_interp = np.copy(data)
        nans, x = nan_helper(data)
        data_interp[nans]= np.interp(x(nans), x(~nans), data[~nans])

        print('RMS voltage:', np.sqrt(np.mean(np.square(data_interp))))
```

```
RMS voltage: 227.28577854695376
```

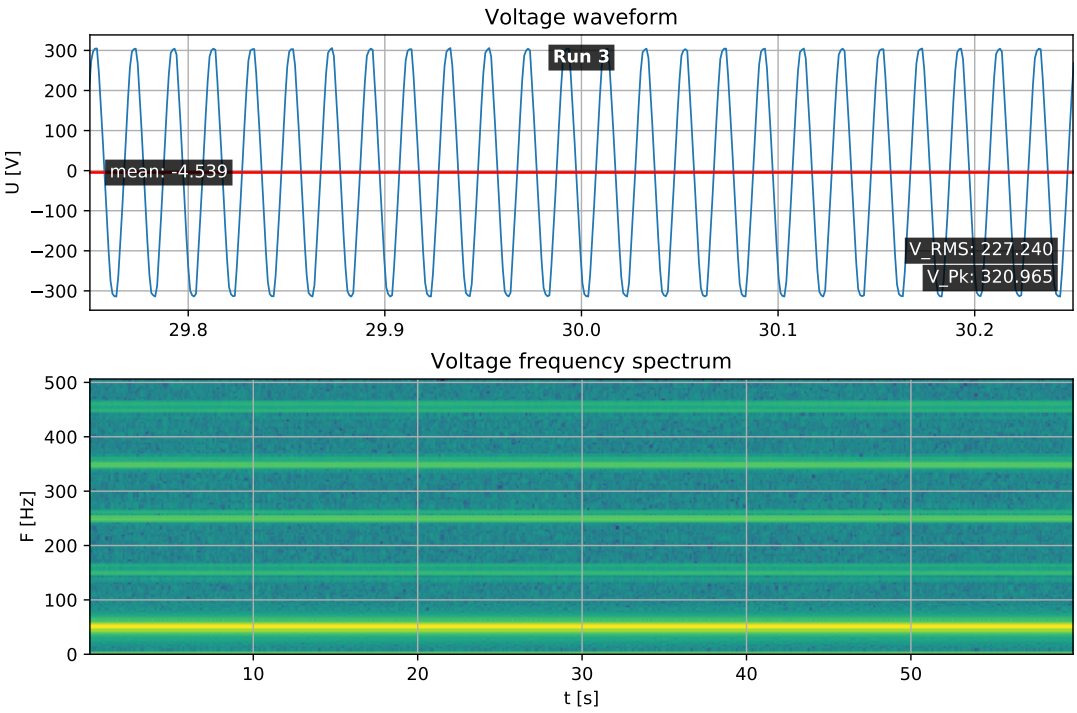
```
In [7]: import itertools
        skip_groups = [ len(list(group))//32 for val, group in itertools.groupby(nans) if val
        print('Number of skipped sample packets:', sum(skip_groups))
        print('Consecutive skipped packets:', ' '.join(f'{val} pkt: {len(list(group))}' for val
```

```
Number of skipped sample packets: 16
Consecutive skipped packets: 1 pkt: 10 2 pkt: 3
```

1.5 Show a preview of loaded data

```
In [8]: fig, (top, bottom) = plt.subplots(2, figsize=(9,6))
        fig.tight_layout(pad=3, h_pad=1.8)

        range_start, range_len = -300, 60 # [s]
```

```
data_slice = data[ int(range_start * sampling_rate) : int((range_start + range_len) * s

top.grid()
top.plot(np.linspace(0, range_len, int(range_len*sampling_rate)), data_slice, lw=1.0)
top.set_xlim([range_len/2-0.25, range_len/2+0.25])
mean = np.mean(data_interp)
rms = np.sqrt(np.mean(np.square(data_interp - mean)))
peak = np.max(np.abs(data_interp - mean))
top.axhline(mean, color='red')
bbox = {'facecolor': 'black', 'alpha': 0.8, 'pad': 2}
top.text(0.02, 0.5, f'mean: {mean:.3f}', transform=top.transAxes, color='white', bbox=b
top.text(0.98, 0.2, f'V_RMS: {rms:.3f}', transform=top.transAxes, color='white', bbox=b
top.text(0.98, 0.1, f'V_Pk: {peak:.3f}', transform=top.transAxes, color='white', bbox=b
top.text(0.5, 0.9, f'Run {run_id}', transform=top.transAxes, color='white', bbox=bbox,

bottom.grid()
bottom.specgram(data_slice, Fs=sampling_rate)
top.set_ylabel('U [V]')
bottom.set_ylabel('F [Hz]')
bottom.set_xlabel('t [s]')

top.set_title('Voltage waveform')
bottom.set_title('Voltage frequency spectrum')
None
```



2 Calculate Short-Time Fourier Transform of capture

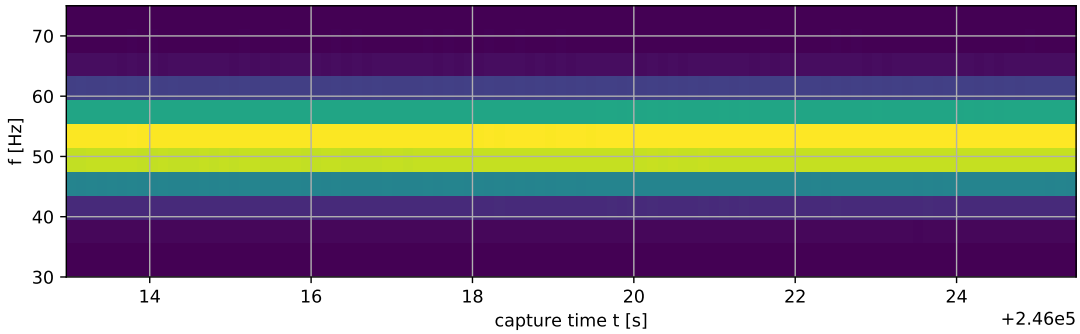
```
In [9]: def calc_stft(data, fs=sampling_rate, ff:'Hz nominal'=50.0):
            analysis_periods = 10
            window_len = 256 # fs * analysis_periods/ff
            nfft_factor = 1
            sigma = window_len/8 # samples

            f, t, Zxx = signal.stft(data,
                        fs = fs,
                        window=('gaussian', sigma),
                        nperseg = window_len,
                        nfft = window_len * nfft_factor)
            print(f'Window length: {window_len:.0f} sp, zero-padded to {window_len * nfft_facto
            stft_output_sampling_rate = 1.0/(t[1] - t[0])
            print('STFT sampling rate:', stft_output_sampling_rate)
            return f, t, Zxx, stft_output_sampling_rate

        f, t, Zxx, stft_output_sampling_rate = calc_stft(data)
```

```
Window length: 256 sp, zero-padded to 256 sp
STFT sampling rate: 7.91015625
```

2.1 Show a preview of STFT results

Cut out our approximate frequency range of interest

```
In [10]: fig, ax = plt.subplots(figsize=(9, 3))
         fig.tight_layout(pad=2, h_pad=0.1)

         ax.pcolormesh(t[-200:-100], f[:250], np.abs(Zxx[:250,-200:-100]))
         ax.set_title(f"Run {last_run}", pad=-20, color='white')
         ax.grid()
         ax.set_ylabel('f [Hz]')
         ax.set_ylim([30, 75]) # Hz
         ax.set_xlabel('capture time t [s]')
         None
```



3 Run Gasior and Gonzalez for precise frequency estimation

Limit analysis to frequency range of interest. If automatic adaption to totally different frequency ranges (e.g. 400Hz) would be necessary, we could switch here based on configuration or a lookup of the STFT bin containing highest overall energy.

As elaborated in the Gasior and Gonzalez Paper [1] the shape of the template function should match the expected peak shape. Peak shape is determined by the STFT window function. As Gasior and Gonzalez note, a gaussian is a very good fit for a steep gaussian window.

```
In [13]: def runner(args):
             frame_f, frame_Z, le_t = args
             # Template function. We use a gaussian here. This function needs to fit the windo
             def gauss(x, *p):
                 A, mu, sigma = p
                 return A*np.exp(-(x-mu)**2/(2.*sigma**2))

             # Calculate initial values for curve fitting
             f_start = frame_f[np.argmax(frame_Z)] # index of strongest bin index
             A_start = np.max(frame_Z) # strongest bin value
             p0 = [A_start, f_start, 1.]
             try:
                 # Fit template to measurement data STFT ROI
                 coeff, var = optimize.curve_fit(gauss, frame_f, frame_Z, p0=p0)
                 _A, rv, _sigma, *_ = coeff # The measured frequency is the mean of the fitted
                 return rv

             except Exception as e:
                 # Handle fit errors
                 return np.nan

         def gasior_gonzalez_fmeas(f, t, Zxx):
             import multiprocessing
```

```
    chunksize = 1000

    n_f, n_t = Zxx.shape
    # Frequency ROI
    f_min, f_max = 30, 70 # Hz
    # Indices of bins within ROI
    bounds_f = slice(np.argmax(f > f_min), np.argmin(f < f_max))

    # Initialize output array
    f_mean = np.zeros(Zxx.shape[1])

    jobs = {}
    with multiprocessing.Pool(multiprocessing.cpu_count()) as pool, tqdm(total=Zxx.sha
        # Iterate over STFT time slices
        for le_t in range(0, Zxx.shape[1], chunksize):
            # Cut out ROI and compute magnitude of complex fourier coefficients
            jobs[le_t] = pool.map_async(runner, [
                    (f[bounds_f], np.abs(Zxx[bounds_f, frame_t]), frame_t) for frame_
                ], callback=lambda _x: tq.update(chunksize))

        pool.close()
        for le_t, future in jobs.items():
            f_mean[le_t:le_t+chunksize] = future.get()
        pool.join()

    # Cut off invalid values at fringes
    return f_mean[1:-2], t[1:-2]

f_mean, f_mean_t = gasior_gonzalez_fmeas(f, t, Zxx)
```

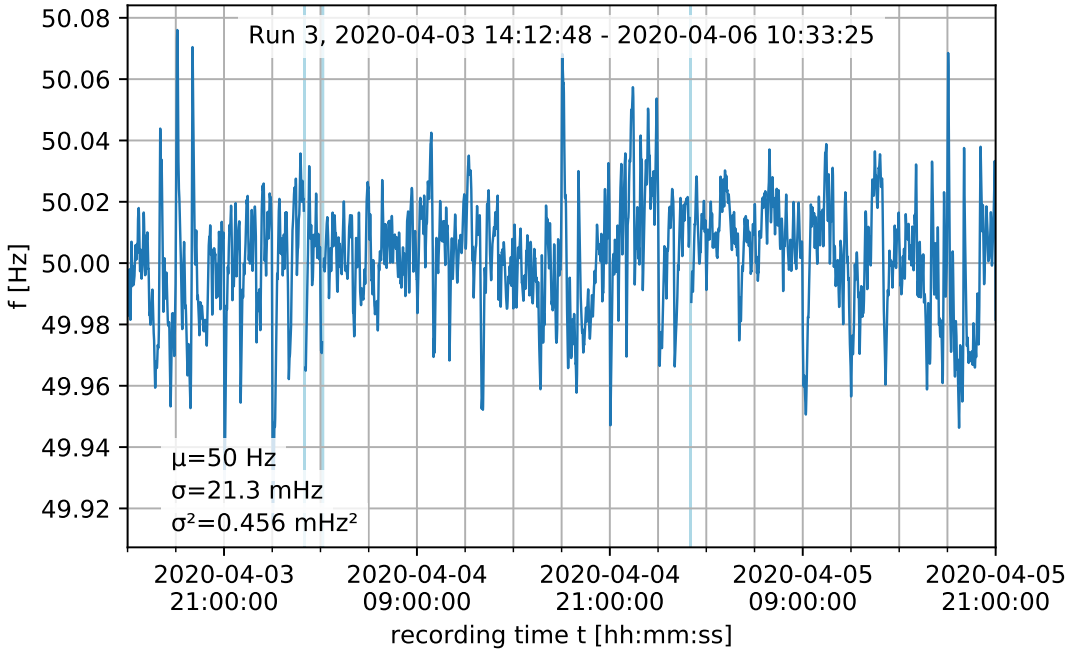
3.1 Produce plots of measurement results

3.1.1 Plot results as time-series data

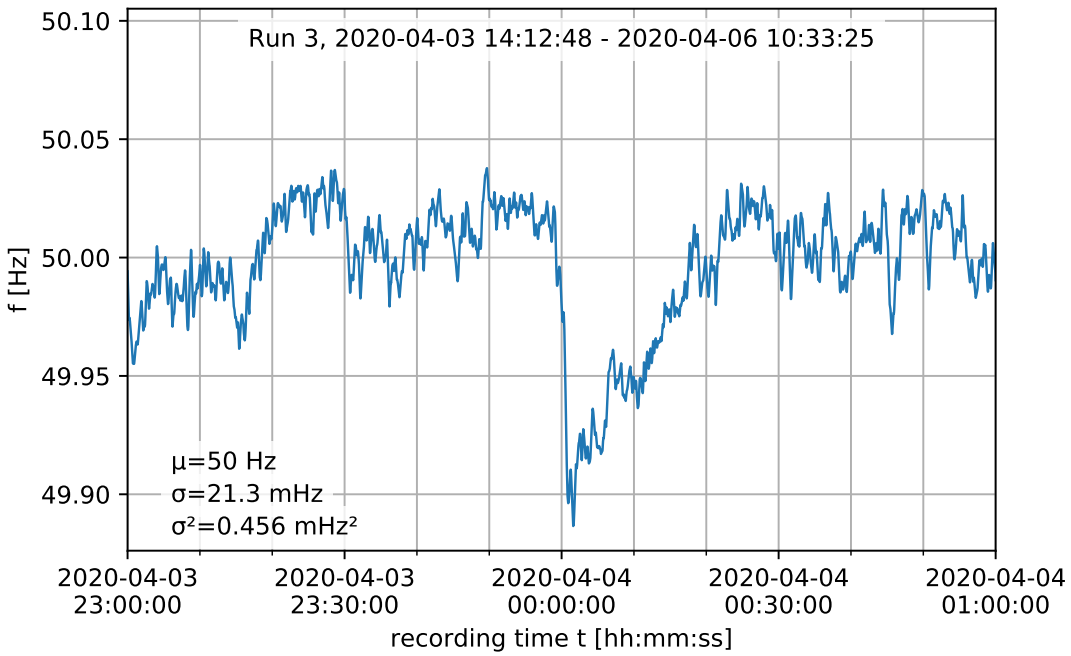
Include measurements of mean, standard deviation and variance of measurement data

```
In [14]: pdate = lambda s: dates.date2num(datetime.datetime.fromisoformat(s))
         td2num = lambda td: dates.date2num(start + datetime.timedelta(seconds=td))

         def plot_freq_trace(outfile, xlim=None, minor_locator=dates.HourLocator(interval=3), m
             fig, ax = plt.subplots(figsize=(6, 4), sharex=True)
             fig.tight_layout(pad=2.2, h_pad=0, w_pad=1)

             if smooth_sec is not None:
                 # smooth data by convolving with a blackman window
                 a = int(10 * smooth_sec)
```

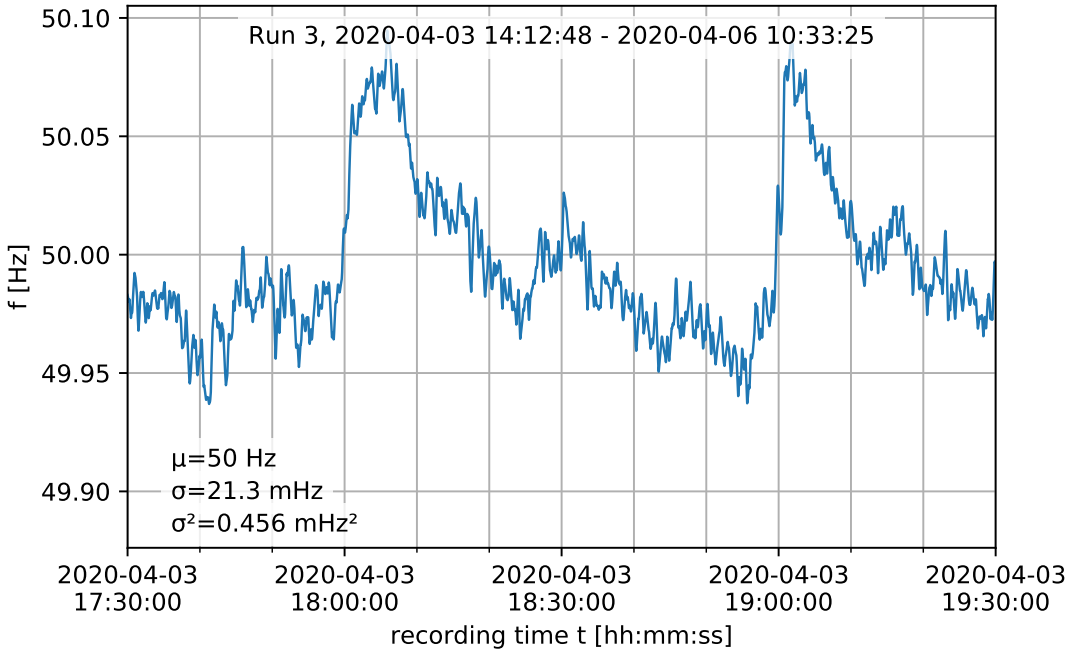
```
            a = a//2*2 + 1 # make a odd
            w = np.blackman(a)
            f_smooth = np.convolve(w/w.sum(), f_mean, mode='valid')
            t_smooth = f_mean_t[a//2:-a//2+1]
        else:
            f_smooth = f_mean
            t_smooth = f_mean_t

        ax.plot([ td2num(td) for td in t_smooth ], f_smooth, lw=1)
        ax.set_ylabel('f [Hz]')

        var = np.var(f_mean[~np.isnan(f_mean)][1:-1])
        mean = np.mean(f_mean[~np.isnan(f_mean)][1:-1])
        ax.text(0.5,  0.93, f'Run {run_id}, {time_window}', transform=ax.transAxes, ha='c
        ax.text(0.05, 0.15, f'={mean:.3g} Hz', transform=ax.transAxes, ha='left', bbox=di
        ax.text(0.05, 0.09, f'={np.sqrt(var) * 1e3:.3g} mHz', transform=ax.transAxes, ha=
        ax.text(0.05, 0.03, f'š={var * 1e3:.3g} mHzš', transform=ax.transAxes, ha='left',

        # Indicate missing values
        for i in np.where(np.isnan(f_mean))[0]:
            ax.axvspan(td2num(t[i]), td2num(t[i+1]), color='lightblue')

        ax.xaxis.set_major_locator(major_locator)
        ax.xaxis.set_minor_locator(minor_locator)
        formatter = dates.DateFormatter('%Y-%m-%d\n%H:%M:%S')
        ax.xaxis.set_major_formatter(formatter)
        ax.set_xlabel('recording time t [hh:mm:ss]')
        ax.grid(True, which='both')
        if xlim is not None:
            ax.set_xlim(xlim)
        fig.savefig(f'fig_out/{outfile}.pdf')
        None

plot_freq_trace('freq_meas_trace_24h',
                xlim=[pdate('2020-04-03 15:00:00'), pdate('2020-04-05 21:00:00')],
                smooth_sec=60*5,
               )
```




```
In [13]: plot_freq_trace('freq_meas_trace_2h_1',
                         xlim=[pdate('2020-04-03 23:00:00'), pdate('2020-04-04 01:00:00')],
                         smooth_sec=10,
                         minor_locator=dates.MinuteLocator(interval=10),
                         major_locator=dates.MinuteLocator(interval=30))
```



```
In [14]: plot_freq_trace('freq_meas_trace_2h_2',
                         xlim=[pdate('2020-04-03 17:30'), pdate('2020-04-03 19:30:00')],
                         smooth_sec=10,
                         minor_locator=dates.MinuteLocator(interval=10),
                         major_locator=dates.MinuteLocator(interval=30))
```



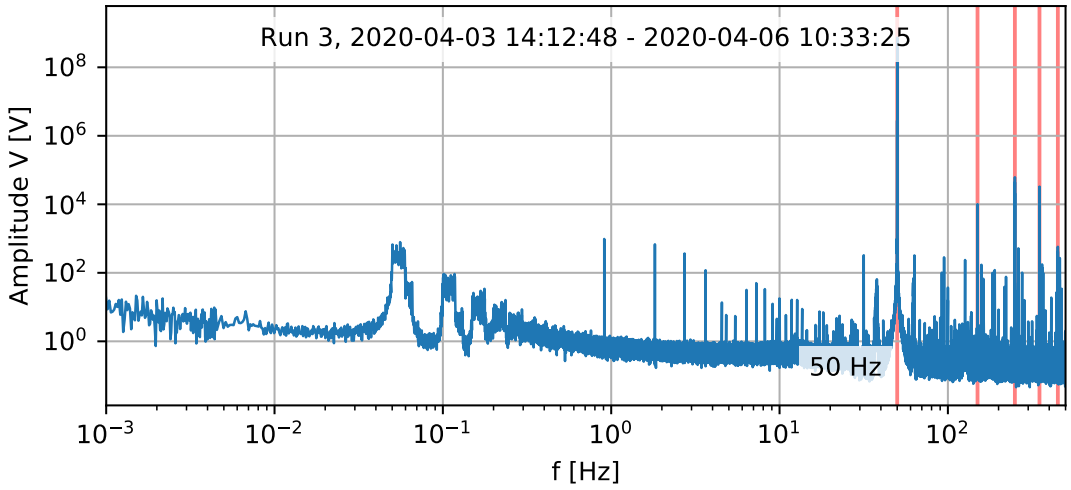
3.1.2 Plot raw mains voltage spectrum

First compute FFT of voltage, then smoothen and plot

```
In [36]: def compute_voltage_fft(data):
             # Number of samplepoints
             N = len(data)
             # sample spacing
             T = 1.0 / sampling_rate
             x = np.linspace(0.0, N*T, N)
             yf = np.absolute(scipy.fftpack.fft(data * sig.blackman(N)))**2
             xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

             yf = 2.0/N * np.abs(yf[:N//2])

             average_from = lambda val, start, average_width: np.hstack([val[:start], [ np.mea
```

```
        average_width = 6
        average_start = 20
        yf = average_from(yf, average_start, average_width)
        xf = average_from(xf, average_start, average_width)
        yf = average_from(yf, 200, average_width)
        xf = average_from(xf, 200, average_width)
        return xf, yf
    #voltage_fft = compute_voltage_fft(data_interp)
```

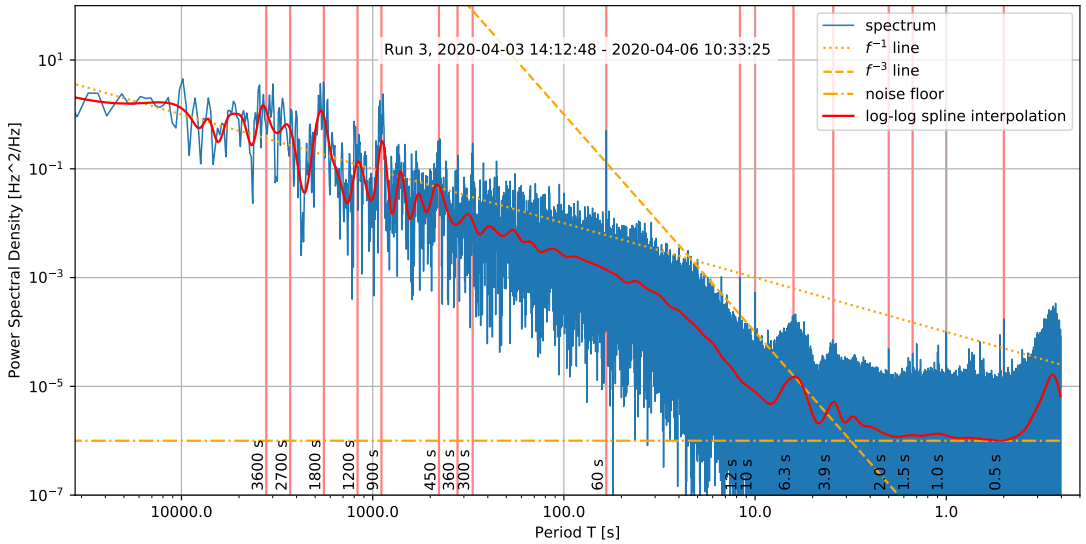
In [37]:
```
def plot_voltage_spectrum(xf, yf):
    fig, ax = plt.subplots(figsize=(6, 3))
    fig.tight_layout()

    yf2 = np.copy(yf)
    #chunk_size = 200
    #for i in range(len(yf)-chunk_size, 1, -chunk_size):
    #    yf2 = np.hstack([yf2[:i], sig.savgol_filter(yf2[i:], 23, 3)])

    ax.loglog(xf, yf2, lw=1)
    #ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, _pos: f'{1/x:.1f}'))
    ax.set_xlabel('f [Hz]')
    ax.set_ylabel('Amplitude V [V]')
    ax.grid()
    ax.set_xlim([0.001, 500])
    fig.subplots_adjust(bottom=0.2)

    for le_f in (50, 150, 250, 350, 450):
        ax.axvline(le_f, color=(1, 0.5, 0.5), zorder=-2)
    ax.annotate('50 Hz', xy=(15, 0.1), xycoords='data', bbox=dict(fc='white', alpha=0
    ax.text(0.5, 0.9, f'Run {run_id}, {time_window}', transform=ax.transAxes, ha='cen
    fig.savefig('fig_out/mains_voltage_spectrum.pdf')
#plot_voltage_spectrum(*voltage_fft)
plot_voltage_spectrum(*compute_voltage_fft(data_interp))
```



3.1.3 Plot frequency spectrum

```
In [33]: def plot_fmeas_spectrum(data):
             # Number of samplepoints
             newcopy = np.copy(data)
             nans, x = nan_helper(newcopy)
             newcopy[nans]= np.interp(x(nans), x(~nans), newcopy[~nans])

             N = len(newcopy)
             # sample spacing
             T = 1.0 / stft_output_sampling_rate
             x = np.linspace(0.0, N*T, N)
             yf = np.absolute(scipy.fftpack.fft(newcopy * sig.blackman(N)))**2
             xf = np.linspace(0.0, stft_output_sampling_rate/2, N//2)

             yf = 2.0/N * np.abs(yf[:N//2])

             #chunk_size = 200
             #for i in range(len(yf)-chunk_size, 1, -chunk_size):
             #    yf = np.hstack([yf[:i], sig.savgol_filter(yf[i:], 23, 3)])

             fig, ax = plt.subplots(figsize=(12,6))
             ax.loglog(xf, yf, label='spectrum', lw=1)
             ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, _pos: f'{1/x:.1f}'))
             ax.set_xlabel('Period T [s]')
             ax.set_ylabel('Power Spectral Density [Hz^2/Hz]')

             for i, t in enumerate([0.5, 1.0, 1.5, 2.0, 3.9, 6.3, 10, 12, 60, 300, 360, 450, 9
                 ax.axvline(1/t, color='red', alpha=0.5, zorder=-1)
```

```
            ax.annotate(f'{t} s', xy=(1/t, 1e-7), xytext=(-10, 5), xycoords='data', textc
        #ax.text(1/60, 10,'60 s', ha='left')
        ax.grid()
        ax.set_xlim([1/(10*3600), 5])
        ax.set_ylim([1e-7, 1e2])
        ax.autoscale(False) # do not include noise illustration lines in autoscaling
        ax.plot(xf[1:], 1e-4/xf[1:], label='$f^{-1}$ line', color='orange', ls=':')
        ax.plot(xf[1:], 1e-8/(xf[1:]**4), label='$f^{-3}$ line', color='orange', ls='--')
        ax.plot(xf[1:], np.tile(1e-6, len(xf)-1), label='noise floor', color='orange', ls=


        #yf2 = np.copy(yf)
        #chunk_size = 50
        #for i in range(len(yf)-chunk_size, 1, -chunk_size):
        #    yf2 = np.hstack([yf2[:i], sig.savgol_filter(yf2[i:], 23, 3)])
        #chunk_size = 2000
        #for i in range(len(yf2)-chunk_size, 1, -chunk_size):
        #    yf2 = np.hstack([yf2[:i], sig.savgol_filter(yf2[i:], 511, 3)])
        #ax.plot(xf, yf2)
        spline_first = 4
        foo = np.log(yf[spline_first:])
        foo_w = np.tile(1, len(foo))
        foo_w[np.isnan(foo)] = 0
        foo[np.isnan(foo)] = 0
        spl = scipy.interpolate.splrep(x=xf[spline_first:], y=foo, w=foo_w, t=np.logspace
        ax.plot(xf[spline_first:], np.exp(scipy.interpolate.splev(xf[spline_first:], spl)

        ax.text(0.5, 0.9, f'Run {run_id}, {time_window}', transform=ax.transAxes, ha='cen

        ax.legend(loc='upper right')
        fig.savefig('fig_out/freq_meas_spectrum.pdf')

        return (xf[spline_first], xf[-1], len(xf[spline_first:])), spl
psd_spl_x, psd_spl = plot_fmeas_spectrum(f_mean)
```



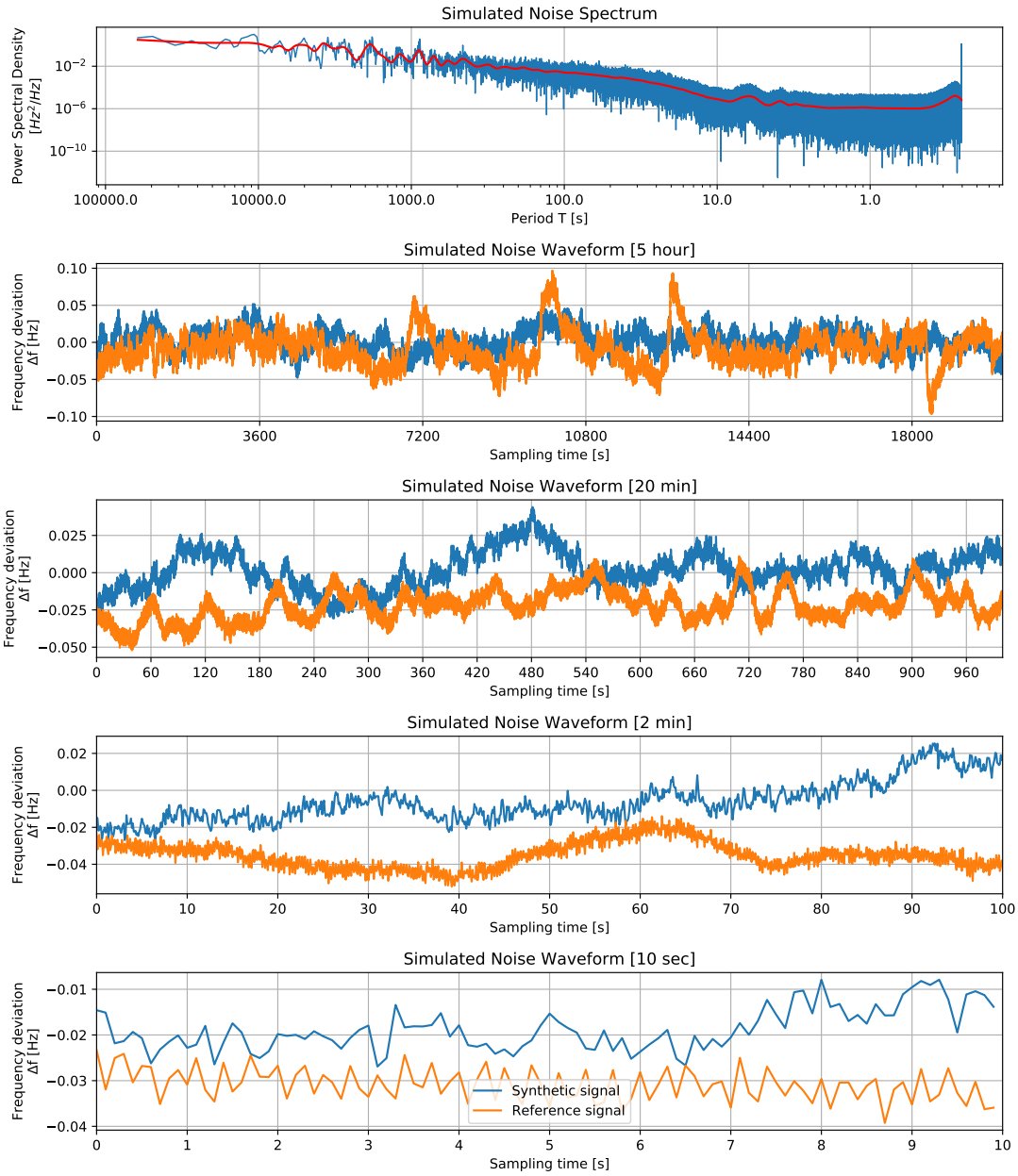
3.2 Export measurement data for modulation simulations

```
In [34]: print(f'Invalid samples: {np.sum(np.isnan(f_mean))} / {len(f_mean)} ({np.sum(np.isnan

         with open(f'data/fmeas_export_ocxo_2day.bin', 'wb') as f:
             for sample in f_mean:
                 if not np.isnan(sample):
                     f.write(struct.pack('<f', sample))
```

```
Invalid samples: 24 / 1946198 (12.3 ppm)
```

```
In [35]: with open(f'grid_freq_psd_spl_{len(psd_spl[1])}pt.json', 'w') as f:
             json.dump({'x_spec': psd_spl_x, 't': psd_spl[0].tolist(), 'c': psd_spl[1].tolist()

         def generate_synthetic_noise(specfile='grid_freq_psd_spl_108pt.json'):
             with open(specfile) as f:
                 d = json.load(f)
                 x = np.linspace(*d['x_spec'])
                 N = len(x)
                 psd_spl = d['t'], d['c'], d['k']

             noise = np.random.normal(size=N) * 2
             spec = scipy.fftpack.fft(noise) **2

             spec *= np.exp(scipy.interpolate.splev(x, psd_spl))

             spec **= 1/2
```

```
    renoise = scipy.fftpack.ifft(spec)
    return x, renoise

def noise_sim(specfile='grid_freq_psd_spl_108pt.json'):
    x, renoise = generate_synthetic_noise(specfile)
    N = len(x)
    respec = 2.0/N * np.absolute(scipy.fftpack.fft(renoise * np.blackman(N))) ** 2

    #xf = np.linspace(0, 10/2, N//2)
    fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(12, 15), gridspec_kw={'

    ax1.loglog(x, respec, lw=1)
    ax1.plot(x, np.exp(scipy.interpolate.splev(x, psd_spl)), color='red', label='log-
    ax1.grid()
    ax1.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, _pos: f'{1/x:.1f}'))
    ax1.set_xlabel('Period T [s]')
    ax1.set_ylabel('Power Spectral Density\n[$Hz^2/Hz$]')
    ax1.set_title('Simulated Noise Spectrum')
    def plot_sigs(ax, lims, tick_ivl, legend_loc=None, title=None):
        ax.plot(renoise[slice(*lims)], label='Synthetic signal')
        ax.plot(f_mean[slice(*lims)] - np.mean(f_mean[~np.isnan(f_mean)]), label='Ref
        ax.grid()
        if legend_loc is not None:
            ax.legend(loc=legend_loc)
        ax.set_ylabel('Frequency deviation\nf [Hz]')
        ax.set_xlabel('Sampling time [s]')
        ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, _pos: f'{x/10:.0f}'))
        ax.xaxis.set_major_locator(plt.MultipleLocator(tick_ivl * 10.0))
        ax.set_xlim([0, lims[1]-lims[0]])
        if title:
            ax.set_title(title)
    plot_sigs(ax2, [10000, 210000], 3600, title='Simulated Noise Waveform [5 hour]')
    plot_sigs(ax3, [10000, 20000], 60, title='Simulated Noise Waveform [20 min]')
    plot_sigs(ax4, [10000, 11000], 10, title='Simulated Noise Waveform [2 min]')
    plot_sigs(ax5, [10000, 10100], 1, legend_loc='lower center', title='Simulated Noi

    return fig

noise_sim().savefig('fig_out/simulated_noise_spectrum.pdf')
```

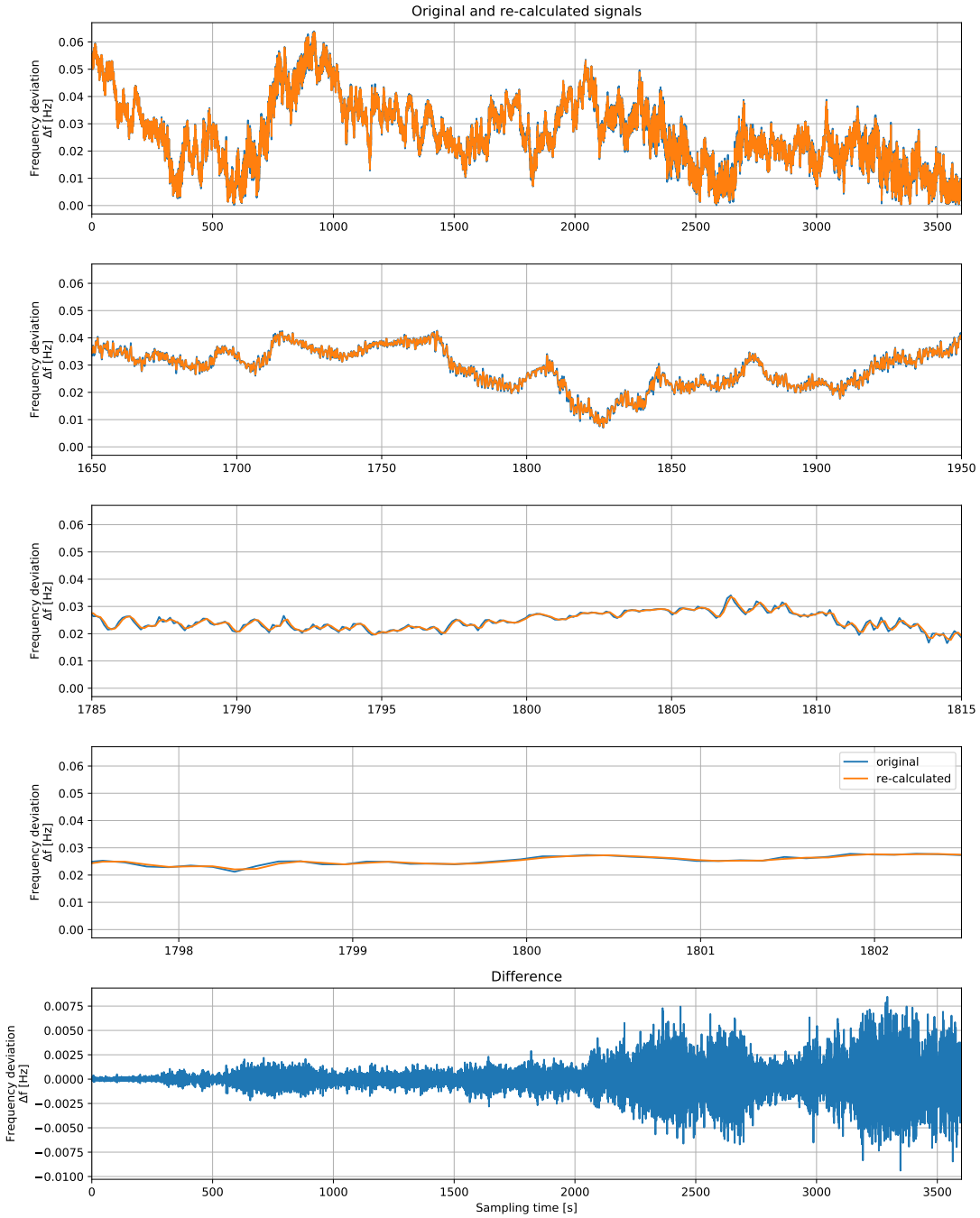



```
In [77]: def do_artificial_noise_simulation(duration:'seconds' = 3600.0, sampling_rate=samplin
             t_pad = 1000.0
             offx = int(t_pad*stft_output_sampling_rate)

             _x, noise_freqs = generate_synthetic_noise()
             noise_freqs = np.absolute(noise_freqs)[offx:][:int(duration * stft_output_samplin
             x = np.linspace(0, duration, int(duration*sampling_rate))
             noise_resampled = np.interp(x, np.linspace(0, len(noise_freqs)/stft_output_sampli
```

```
        phase_acc = 0.0
        out = np.zeros(len(noise_resampled))
        for i, f in enumerate(noise_resampled):
            phase_acc += 2*np.pi*(50.0 + f) / sampling_rate
            out[i] = np.sin(phase_acc)
            if phase_acc > 2*np.pi:
                phase_acc -= 2*np.pi

        return out, noise_freqs

    def recalc_f(duration=3600.0):
        new_data, orig_noise = do_artificial_noise_simulation(duration, sampling_rate)
        f, t, Zxx, stft_output_sampling_rate = calc_stft(new_data, sampling_rate)
        #plt.matshow(np.absolute(Zxx), aspect='auto')
        f_mean, t = gasior_gonzalez_fmeas(f, t, Zxx)
        return t, f_mean, orig_noise

    def feedback_plot(duration=3600):
        fig, axs = plt.subplots(5, 1, figsize=(12, 15))
        new_t, new_mean, orig_noise = recalc_f()
        for ax, time_range in zip(axs.flatten(), (duration, 300, 30, 5)):
            ax.plot(new_t, orig_noise[1:-1], label='original')
            ax.plot(new_t, new_mean - 50, label='re-calculated')
            ax.grid()
            ax.set_xlim((duration/2-time_range/2, duration/2+time_range/2))
            ax.set_ylabel('Frequency deviation\nf [Hz]')
        axs[-2].legend()
        delta = orig_noise[1:-1] - (new_mean - 50)
        print(np.sqrt(np.mean(np.square(delta))))
        axs[-1].plot(new_t, delta)
        axs[-1].set_xlabel('Sampling time [s]')
        axs[-1].set_xlim((0, duration))
        axs[-1].grid()
        axs[-1].set_title('Difference')
        axs[-1].set_ylabel('Frequency deviation\nf [Hz]')
        axs[0].set_title('Original and re-calculated signals')
        fig.tight_layout()
        return fig

    feedback_plot().savefig('fig_out/freq_meas_feedback.pdf')
```

```
Window length: 256 sp, zero-padded to 256 sp
STFT sampling rate: 7.91015625
```

```
0.0012170400234126895
```



4 References

1. **Gasior, M. & Gonzalez, J.** Improving FFT frequency measurement resolution by parabolic and gaussian interpolation *CERN-AB-Note-2004-021, CERN-AB-Note-2004-021, 2004*

A.2 Grid frequency estimation validation against ROCOF test suite

In [17]:
```
import math
import struct

import numpy as np
from scipy import signal, optimize
from matplotlib import pyplot as plt

import rocof_test_data
```

In [18]:
```
import matplotlib
from IPython.display import set_matplotlib_formats
#%matplotlib widget
%matplotlib inline
set_matplotlib_formats('png', 'pdf')
font = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 10}
matplotlib.rc('font', **font)
```

In [19]:
```
fs = 1000 # Hz
ff = 50 # Hz
duration = 60 # seconds
# test_data = rocof_test_data.sample_waveform(rocof_test_data.test_close_interharmoni
#                                             duration=20,
#                                             sampling_rate=fs,
#                                             frequency=ff)[0]
# test_data = rocof_test_data.sample_waveform(rocof_test_data.gen_noise(fmin=10, ampl
#                                             duration=20,
#                                             sampling_rate=fs,
#                                             frequency=ff)[0]

test_data = []
test_labels = [ fun.__name__.replace('test_', '') for fun in rocof_test_data.all_tests
for gen in rocof_test_data.all_tests:
    test_data.append(rocof_test_data.sample_waveform(gen(),
                                                     duration=duration,
                                                     sampling_rate=fs,
                                                     frequency=ff)[0])
# d = 10 # seconds
# test_data = np.sin(2*np.pi * ff * np.linspace(0, d, int(d*fs)))
```

In [20]:
```
spr_fmt = f'{fs}Hz' if fs<1000 else f'{fs/1e3:f}'.rstrip('.0') + 'kHz'
for label, data in zip(test_labels, test_data):
    with open(f'rocof_test_data/rocof_test_{label}_{spr_fmt}.bin', 'wb') as f:
```

```
                for sample in data:
                    f.write(struct.pack('<f', sample))
```

In [21]:
```
analysis_periods = 10
window_len = 256 # fs * analysis_periods/ff
nfft_factor = 1
sigma = window_len/8 # samples
quantization_bits = 14

ffts = []
for item in test_data:
    f, t, Zxx = signal.stft((item * (2**(quantization_bits-1) - 1)).round().astype(np
                fs = fs,
                window=('gaussian', sigma),
                nperseg = window_len,
                nfft = window_len * nfft_factor)
                #boundary = 'zeros')
    ffts.append((f, t, Zxx))
```

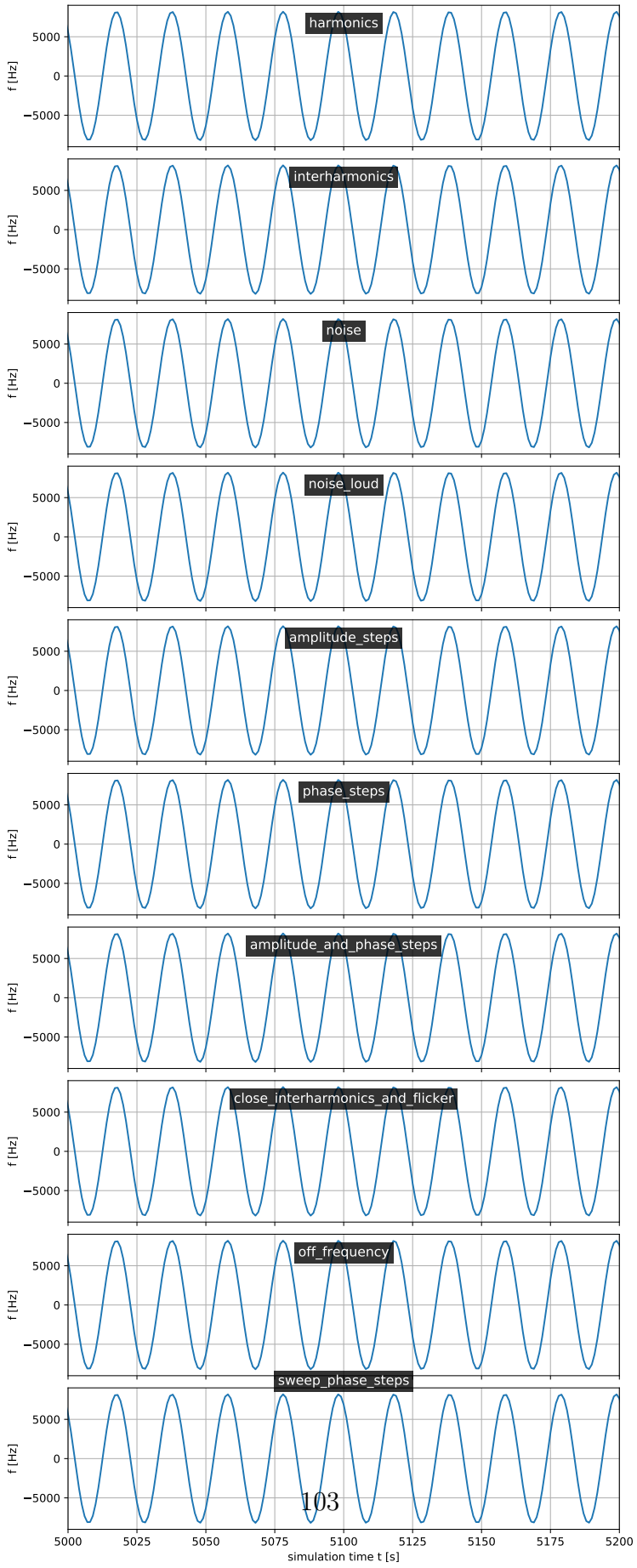
In [22]:
```
Zxx.shape
```

Out[22]:
```
(129, 470)
```

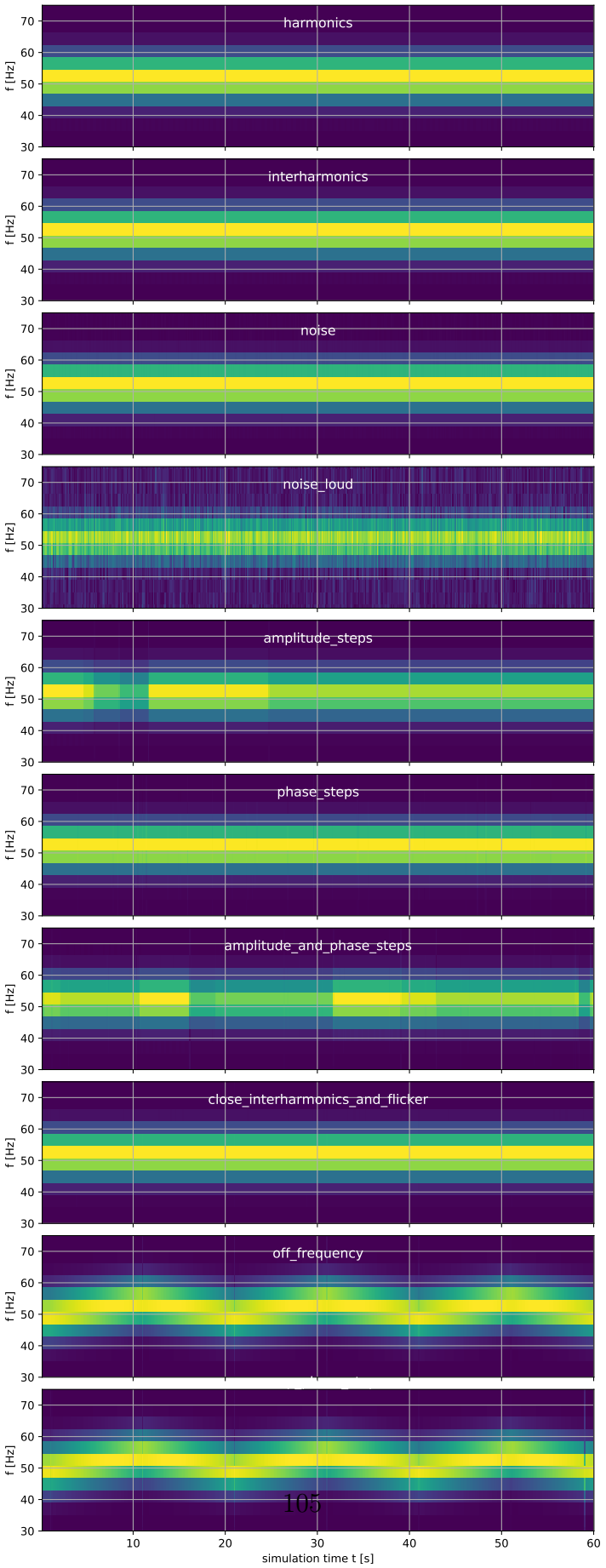
In [23]:
```
1000/256
```

Out[23]:
```
3.90625
```

In [24]:
```
fig, ax = plt.subplots(len(test_data), figsize=(8, 20), sharex=True)
fig.tight_layout(pad=2, h_pad=0.1)

for fft, ax, label in zip(test_data, ax.flatten(), test_labels):
    ax.plot((item * (2**(quantization_bits-1) - 1)).round())

    ax.set_title(label, pad=-20, color='white', bbox=dict(boxstyle="square", ec=(0,0,
    ax.grid()
    ax.set_ylabel('f [Hz]')
ax.set_xlabel('simulation time t [s]')
ax.set_xlim([5000, 5200])
None
```


harmonics
interharmonics
noise
noise_loud
amplitude_steps
phase_steps
amplitude_and_phase_steps
close_interharmonics_and_flicker
off_frequency
sweep_phase_steps
f [Hz]
5000
0
−5000
5000
5025
5050
5075
5100
5125
5150
5175
5200
simulation time t [s]

In [25]:
```
fig, ax = plt.subplots(len(test_data), figsize=(8, 20), sharex=True)
fig.tight_layout(pad=2, h_pad=0.1)

for fft, ax, label in zip(ffts, ax.flatten(), test_labels):
    f, t, Zxx = fft
    ax.pcolormesh(t[1:], f[:250], np.abs(Zxx[:250,1:]))
    ax.set_title(label, pad=-20, color='white')
    ax.grid()
    ax.set_ylabel('f [Hz]')
    ax.set_ylim([30, 75]) # Hz
ax.set_xlabel('simulation time t [s]')
None
```

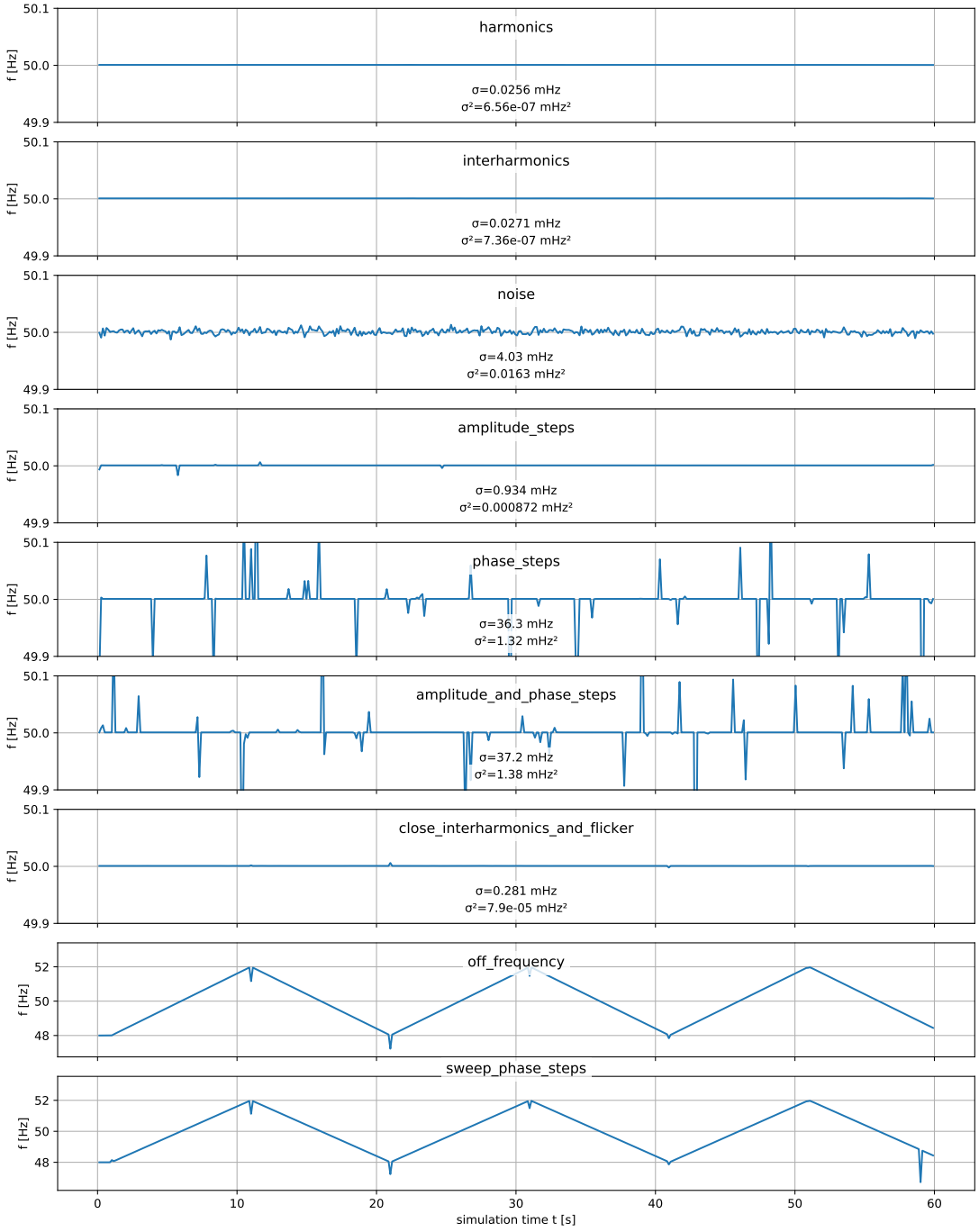

harmonics
interharmonics
noise
noise_loud
amplitude_steps
phase_steps
amplitude_and_phase_steps
close_interharmonics_and_flicker
off_frequency
f [Hz]
30
40
50
60
70
10
20
30
40
50
60
simulation time t [s]

In [26]: f

Out[26]:
```
array([  0.     ,   3.90625,   7.8125 ,  11.71875,  15.625  ,  19.53125,
        23.4375 ,  27.34375,  31.25   ,  35.15625,  39.0625 ,  42.96875,
        46.875  ,  50.78125,  54.6875 ,  58.59375,  62.5    ,  66.40625,
        70.3125 ,  74.21875,  78.125  ,  82.03125,  85.9375 ,  89.84375,
        93.75   ,  97.65625, 101.5625 , 105.46875, 109.375  , 113.28125,
       117.1875 , 121.09375, 125.     , 128.90625, 132.8125 , 136.71875,
       140.625  , 144.53125, 148.4375 , 152.34375, 156.25   , 160.15625,
       164.0625 , 167.96875, 171.875  , 175.78125, 179.6875 , 183.59375,
       187.5    , 191.40625, 195.3125 , 199.21875, 203.125  , 207.03125,
       210.9375 , 214.84375, 218.75   , 222.65625, 226.5625 , 230.46875,
       234.375  , 238.28125, 242.1875 , 246.09375, 250.     , 253.90625,
       257.8125 , 261.71875, 265.625  , 269.53125, 273.4375 , 277.34375,
       281.25   , 285.15625, 289.0625 , 292.96875, 296.875  , 300.78125,
       304.6875 , 308.59375, 312.5    , 316.40625, 320.3125 , 324.21875,
       328.125  , 332.03125, 335.9375 , 339.84375, 343.75   , 347.65625,
       351.5625 , 355.46875, 359.375  , 363.28125, 367.1875 , 371.09375,
       375.     , 378.90625, 382.8125 , 386.71875, 390.625  , 394.53125,
       398.4375 , 402.34375, 406.25   , 410.15625, 414.0625 , 417.96875,
       421.875  , 425.78125, 429.6875 , 433.59375, 437.5    , 441.40625,
       445.3125 , 449.21875, 453.125  , 457.03125, 460.9375 , 464.84375,
       468.75   , 472.65625, 476.5625 , 480.46875, 484.375  , 488.28125,
       492.1875 , 496.09375, 500.     ])
```

In [35]:
```
fig, axs = plt.subplots(len(test_data)-1, figsize=(12, 15), sharex=True)
axs = axs.flatten()

for fft, label in zip(ffts, test_labels):
    if label in ['noise_loud']: # custom test case, not part of upstream suite
        continue
    ax, *axs = axs

    f, f_t, Zxx = fft

    n_f, n_t = Zxx.shape
    f_min, f_max = 30, 70 # Hz
    bounds_f = slice(np.argmax(f > f_min), np.argmin(f < f_max))

    f_mean = np.zeros(Zxx.shape[1])
    for t in range(1, Zxx.shape[1] - 1):
        frame_f = f[bounds_f]
        frame_step = frame_f[1] - frame_f[0]
        time_step = f_t[1] - f_t[0]
        frame_Z = np.abs(Zxx[bounds_f, t])
```

```
            def gauss(x, *p):
                A, mu, sigma = p
                return A*np.exp(-(x-mu)**2/(2.*sigma**2))

            f_start = frame_f[np.argmax(frame_Z)]
            A_start = np.max(frame_Z)
            p0 = [A_start, f_start, 1.]
            try:
                coeff, var = optimize.curve_fit(gauss, frame_f, frame_Z, p0=p0)
                A, mu, sigma, *_ = coeff
                f_mean[t] = mu
            except RuntimeError:
                f_mean[t] = np.nan
        ax.plot(f_t[1:-1], f_mean[1:-1])

        ax.set_title(label, pad=-20, bbox=dict(fc='white', alpha=0.8, ec='none'))
        ax.set_ylabel('f [Hz]')
        ax.grid()
        if not label in ['off_frequency', 'sweep_phase_steps']:
            ax.set_ylim([49.90, 50.10])
            var = np.var(f_mean[1:-1])
            ax.text(0.5, 0.1, f'š={var * 1e3:.3g} mHzš', transform=ax.transAxes, ha='cent
            ax.text(0.5, 0.25, f'={np.sqrt(var) * 1e3:.3g} mHz', transform=ax.transAxes,
        else:
            f_min, f_max = min(f_mean[1:-1]), max(f_mean[1:-1])
            delta = f_max - f_min
            ax.set_ylim(f_min - delta * 0.1, f_max + delta * 0.3)

    ax.set_xlabel('simulation time t [s]')
    fig.tight_layout(pad=2.2, h_pad=0, w_pad=1)
    fig.savefig('fig_out/freq_meas_rocof_reference.pdf')
    None
```


harmonics
σ=0.0256 mHz
σ²=6.56e-07 mHz²
interharmonics
σ=0.0271 mHz
σ²=7.36e-07 mHz²
noise
σ=4.03 mHz
σ²=0.0163 mHz²
amplitude_steps
σ=0.934 mHz
σ²=0.000872 mHz²
phase_steps
σ=36.3 mHz
σ²=1.32 mHz²
amplitude_and_phase_steps
σ=37.2 mHz
σ²=1.38 mHz²
close_interharmonics_and_flicker
σ=0.281 mHz
σ²=7.9e-05 mHz²
off_frequency
sweep_phase_steps
f [Hz]
simulation time t [s]

A.3 Frequency sensor clock stability analysis

1 Setup

1.1 Import required packages

```
In [7]: import math
        import sqlite3
        import struct
        import datetime
        import scipy.fftpack
        from scipy import signal as sig
        from scipy import optimize as opt

        import matplotlib
        from matplotlib import pyplot as plt
        from matplotlib import patches
        import numpy as np
        from scipy import signal, optimize
        from tqdm.notebook import tnrange, tqdm
        from IPython.display import set_matplotlib_formats
```

```
In [2]: %matplotlib inline
        set_matplotlib_formats('png', 'pdf')
```

1.2 Load data series information from sqlite capture file

One capture file may contain multiple runs/data series. Display a list of runs and their start/end time and sample count, then select the newest one in `last_run` variable.

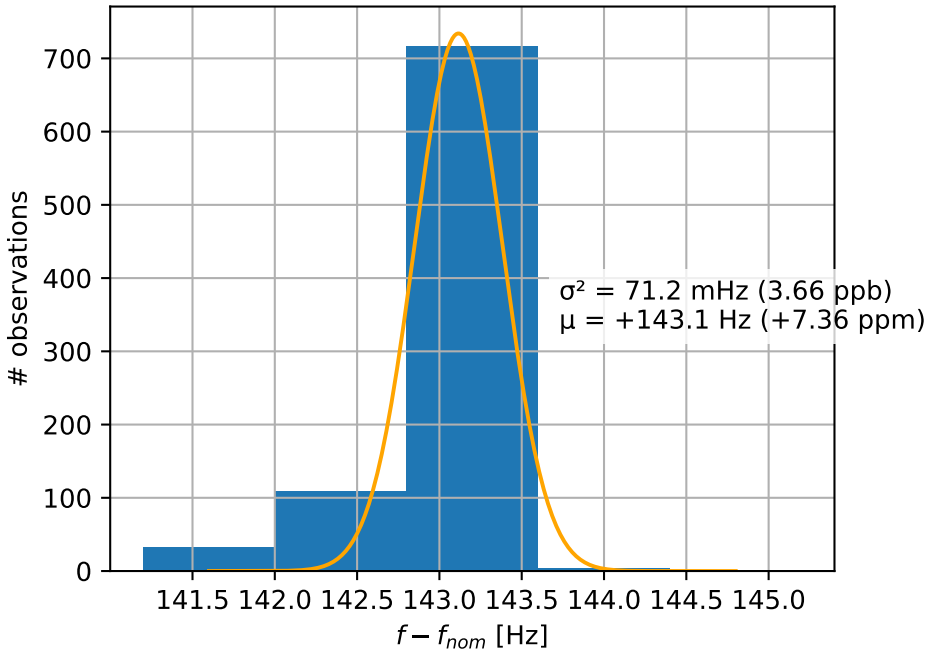
```
In [3]: db = sqlite3.connect('data/waveform_1pps_debug.sqlite3')
```

```
In [4]: for run_id, start, end, count in db.execute('SELECT run_id, MIN(rx_ts), MAX(rx_ts), COU
            foo = lambda x: datetime.datetime.fromtimestamp(x/1000)
            start, end = foo(start), foo(end)
            print(f'Run {run_id:03d}: {start:%Y-%m-%d %H:%M:%S} - {end:%Y-%m-%d %H:%M:%S} ({st
        last_run, n_records = run_id, count
        sampling_rate = 1000.0
```

```
Run 000: 2020-03-31 16:58:00 - 2020-03-31 16:58:36 (  0:00:36.029,     36512sp)
Run 001: 2020-03-31 16:58:51 - 2020-03-31 17:05:19 (  0:06:27.729,    392608sp)
Run 002: 2020-03-31 17:07:02 - 2020-03-31 17:41:34 (  0:34:32.105,     37024sp)
Run 003: 2020-03-31 18:50:05 - 2020-03-31 18:50:43 (  0:00:37.576,     38048sp)
Run 004: 2020-03-31 18:54:08 - 2020-03-31 19:14:32 (  0:20:24.104,   1239424sp)
```

2 Calculate period measurement histograms and convert to Hz

In [9]:
```
histogram = np.array(db.execute('SELECT gps_1pps, COUNT(*) FROM measurements WHERE gps_
hist_plot = histogram.astype(float)[1:-1]
hist_plot[:, 0] *= 2 / 5 * 2
hist_plot[:, 1] /= (1000 / 32)

f_nom = 19.440e6

font = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 10}
matplotlib.rc('font', **font)
fig, ax = plt.subplots(figsize=(5, 4))
ax.grid()
# We have a bug that causes our measurements to occassionally be out by +/- 65534 coun
# For now, fix this by simply throwing away these (very obviously invalid) bins.
ax.bar(hist_plot[:,0] - f_nom , hist_plot[:, 1])

def gauss(x, *p):
    A, mu, sigma = p
    return A*np.exp(-(x-mu)**2/(2.*sigma**2))

gauss_x = np.linspace(np.min(hist_plot[:,0]), np.max(hist_plot[:,0]), 10000)
coeff, var_matrix = opt.curve_fit(gauss, hist_plot[:,0], hist_plot[:,1], p0=[np.max(his
hist_fit = gauss(gauss_x, *coeff)
ax.plot(gauss_x - f_nom, hist_fit, color='orange')
_A, mu, sigma = coeff
bbox_props = dict(fc='white', alpha=0.8, ec='none')
ax.annotate(f'š = {sigma**2 * 1e3:.1f} mHz ({sigma**2 / f_nom * 1e9:.2f} ppb)\n'
            f' = {mu-f_nom:+.1f} Hz ({(mu-f_nom)/f_nom * 1e6:+.2f} ppm)',
            xy=[0.6, 0.5], xycoords='figure fraction', bbox=bbox_props)
ax.set_xlabel('$f - f_{nom}$ [Hz]')
ax.set_ylabel('# observations')

#ax.set_title('OCXO frequency derivation relative to GPS 1pps')
fig.savefig('fig_out/ocxo_freq_stability.pdf', format='pdf')
```


700
600
500
400
300
200
100
0
observations
σ² = 71.2 mHz (3.66 ppb)
μ = +143.1 Hz (+7.36 ppm)
141.5 142.0 142.5 143.0 143.5 144.0 144.5 145.0
$f - f_{nom}$ [Hz]

A.4 DSSS modulation experiments

1 Setup

1.1 Import required packages

In [2]:
```
import struct
import random
import itertools
import datetime
import multiprocessing
from collections import defaultdict
import json
import traceback
import glob

from matplotlib import pyplot as plt
import matplotlib
from matplotlib import ticker
import numpy as np
from scipy import signal as sig
from scipy import fftpack as fftpack
import ipywidgets
from IPython.display import set_matplotlib_formats

from tqdm.notebook import tqdm
import colorednoise

np.set_printoptions(linewidth=240)
```

In [3]:
```
#%matplotlib widget
%matplotlib inline
set_matplotlib_formats('png', 'pdf')
font = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 6}
matplotlib.rc('font', **font)
```

1.2 Define mains frequency sampling rate

This is the rate of mains frequency measurements, also called “reporting rate”.

In [4]:
```
sampling_rate = 10 # sp/s
```

1.3 Library functions

1.3.1 Gold code generator

```
In [5]: # From https://github.com/mubeta06/python/blob/master/signal_processing/sp/gold.py
        preferred_pairs = {5:[[2],[1,2,3]], 6:[[5],[1,4,5]], 7:[[4],[4,5,6]],
                           8:[[1,2,3,6,7],[1,2,7]], 9:[[5],[3,5,6]],
                           10:[[2,5,9],[3,4,6,8,9]], 11:[[9],[3,6,9]]}

        def gen_gold(seq1, seq2):
            gold = [seq1, seq2]
            for shift in range(len(seq1)):
                gold.append(seq1 ^ np.roll(seq2, -shift))
            return gold

        def gold(n):
            n = int(n)
            if not n in preferred_pairs:
                raise KeyError('preferred pairs for %s bits unknown' % str(n))
            t0, t1 = preferred_pairs[n]
            (seq0, _st0), (seq1, _st1) = sig.max_len_seq(n, taps=t0), sig.max_len_seq(n, taps=
            return gen_gold(seq0, seq1)
```

1.3.2 Gold code modulator

```
In [6]: def modulate(data, nbits=5, pad=True):
            # 0, 1 -> -1, 1
            mask = np.array(gold(nbits))*2 - 1

            sel = mask[data>>1]
            data_lsb_centered = ((data&1)*2 - 1)

            signal = (np.multiply(sel, np.tile(data_lsb_centered, (2**nbits-1, 1))).T).flatten(
            if pad:
                return np.hstack([ np.zeros(len(mask)), signal, np.zeros(len(mask)) ])
            else:
                return signal
```

1.3.3 Gold code correlator

This function, used by the prototype demodulation algorithm below, correlates a signal against all 2^n+1 Gold sequences. Given an input signal of length k it produces an output matrix of dimensions $(2^n + 1, k)$ with one column for each shift of the reference Gold sequences w.r.t. the input signal and one row per Gold sequence.

```
In [7]: def correlate(sequence, nbits=5, decimation=1, mask_filter=lambda x: x):
            mask = np.tile(np.array(gold(nbits))[:,:,np.newaxis]*2 - 1, (1, 1, decimation)).re

            # Our input signal has large DC bias. Remove DC bias to reduce numerical errors du
```

```
    sequence -= np.mean(sequence)

    return np.array([np.correlate(sequence, row, mode='full') for row in mask])
```

1.3.4 Read recorded mains frequency data from exported capture file

```
In [8]: with open('data/fmeas_export_ocxo_2day.bin', 'rb') as f:
            meas_data = np.copy(np.frombuffer(f.read(), dtype='float32'))
            print('mean:', np.mean(meas_data), 'len:', len(meas_data))
            meas_data -= np.mean(meas_data)
            def mains_noise(n):
                last_valid = len(meas_data) - n
                start = np.random.randint(last_valid)
                return meas_data[start:start+n]
```

```
mean: 50.00341 len: 1946174
```

1.3.5 Test signal generator

This generates deterministically random test data, modulates it using the Gold code modulator, scales it to a given target amplitude and adds noise from recorded data above.

```
In [9]: def generate_test_signal(duration, nbits=6, signal_amplitude=2.0e-3, decimation=10, se
            test_data = np.random.RandomState(seed=seed).randint(0, 2 * (2**nbits), duration)

            signal = np.repeat(modulate(test_data, nbits) * 2.0 - 1, decimation) * signal_ampli
            noise = mains_noise(len(signal))

            return test_data, signal + noise
```

1.4 Signal exporters for hardware testing

The following two functions generate test data to test the firmware implementation in software simulations.

```
In [10]: def do_export_clean():
             test_duration = 200
             test_nbits = 5
             test_signal_amplitude=2.0e-3
             test_decimation=10

             for test_signal_amplitude in [2.0e-3, 20e-3, 200e-3, 2]:
                 test_data = np.random.RandomState(seed=0).randint(0, 2 * (2**test_nbits), tes
                 #test_data = np.array([0, 1, 2, 3] * 50)
                 signal = np.repeat(modulate(test_data, test_nbits, pad=False) * 2.0 - 1, test
                 with open(f'dsss_test_signals/dsss_test_noiseless_{test_signal_amplitude*1000
                     for e in signal:
                         f.write(struct.pack('<f', e))
```

```
In [11]: def do_export_noisy():
             test_duration = 32
             test_nbits = 5
             test_signal_amplitude=2.0e-3
             test_decimation=10
             test_signal_amplitude = 200e-3
             noise_level = 10e-3

             #test_data = np.random.RandomState(seed=0).randint(0, 2 * (2**test_nbits), test_d
             #test_data = np.array([0, 1, 2, 3] * 50)
             test_data = np.array(range(test_duration))
             signal = np.repeat(modulate(test_data, test_nbits, pad=False) * 2.0 - 1, test_dec
             noise = colorednoise.powerlaw_psd_gaussian(1, len(signal)*10) * noise_level
             noise[-int(1.5*len(signal)):][:len(signal)] += signal

             with open(f'dsss_test_signals/dsss_test_noisy_padded.bin', 'wb') as f:
                 for e in noise:
                     f.write(struct.pack('<f', e))
```

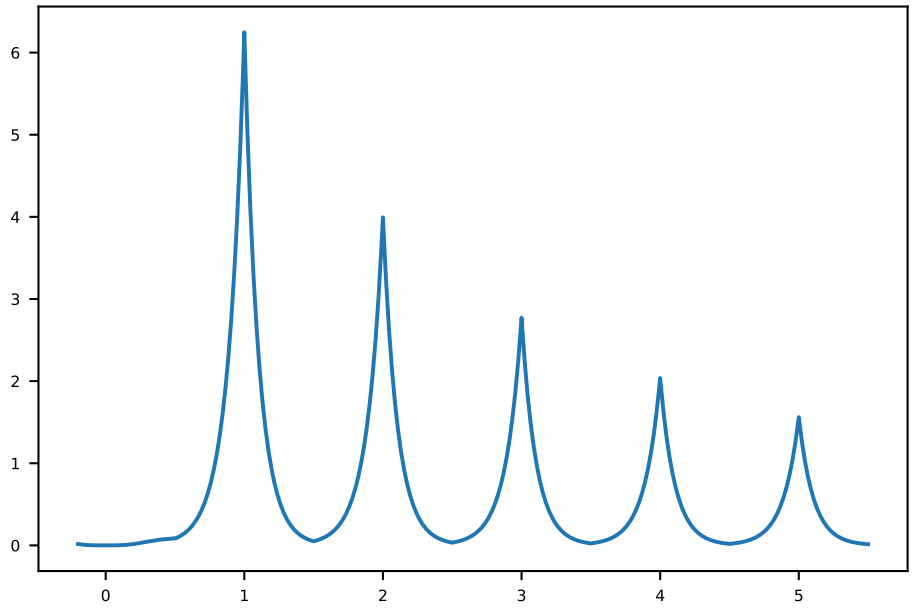
2 The algorithm

2.1 First we define some components used in our algorithm.

The following function is used to score a new correlation peak against previous peaks. The aim is to assign a high fitness the closer the peak lies to a multiple of one symbol period from the last peak. The first peak is the ideal case, subsequent peaks correspond to dropped symbols.

```
In [12]: nonlinear_distance = lambda x: 100**(2*np.abs(0.5-x%1)) / (np.abs(x)+3)**2 * (np.clip

         def plot_distance_func():
             fig, ax = plt.subplots()
             x = np.linspace(-0.2, 5.5, 10000)
             ax.plot(x, nonlinear_distance(x))
```

```
In [13]: plot_distance_func()
```



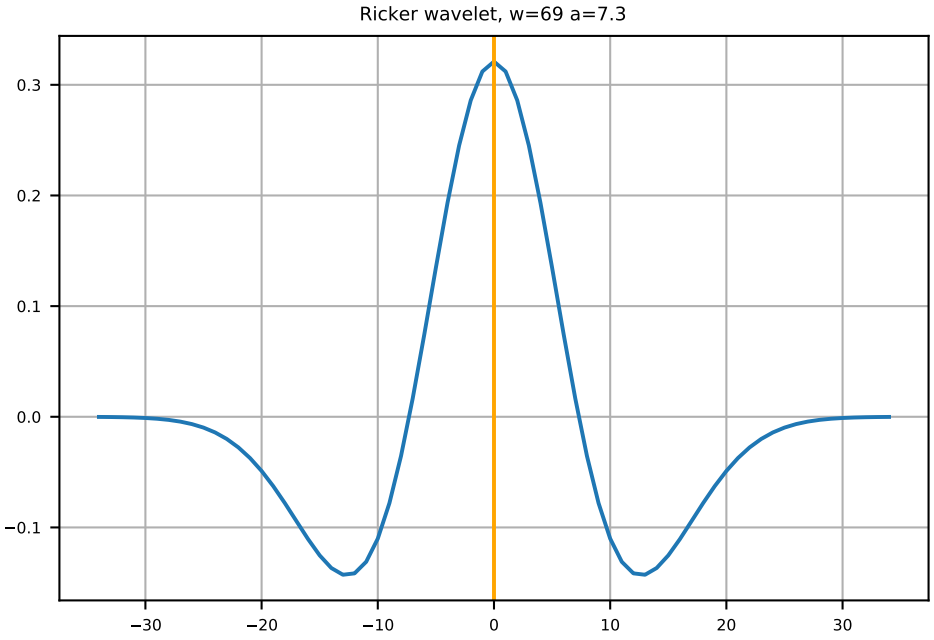
2.2 Ricker wavelet computations for firmware implementation

For our firmware implementation we need a ricker wavelet lookup table. To find out the size of this lookup table, we calculate the truncation error for a given size below.

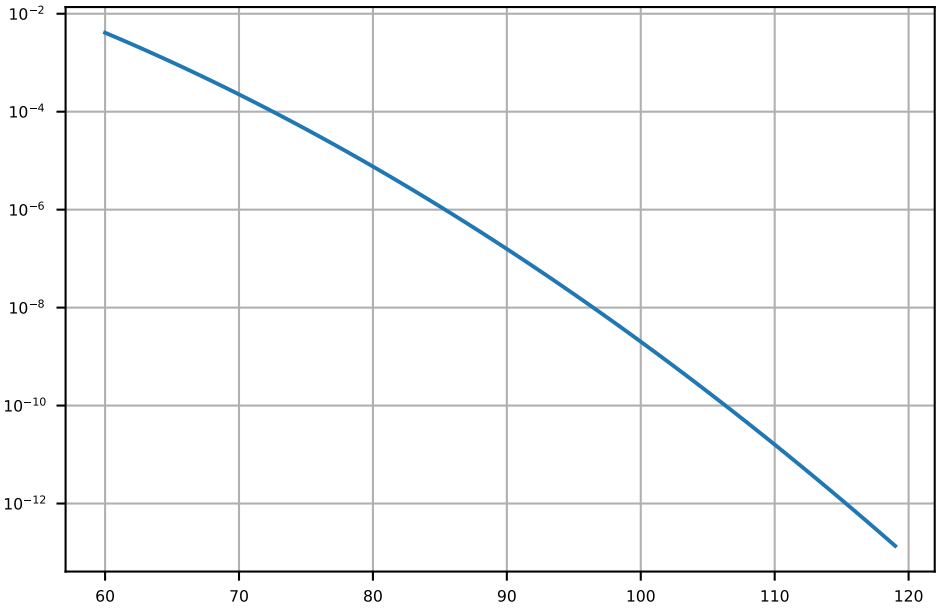
```
In [14]: noprint = lambda *args, **kwargs: None
```

```
In [15]: fig, ax = plt.subplots()
         w = 69
         a = 7.3
         ax.plot(range(-w//2+1, w//2+1), sig.ricker(w, a))
         ax.grid()
         ax.axvline(0, color='orange')
         ax.set_title(f'Ricker wavelet, w={w} a={a}')
```

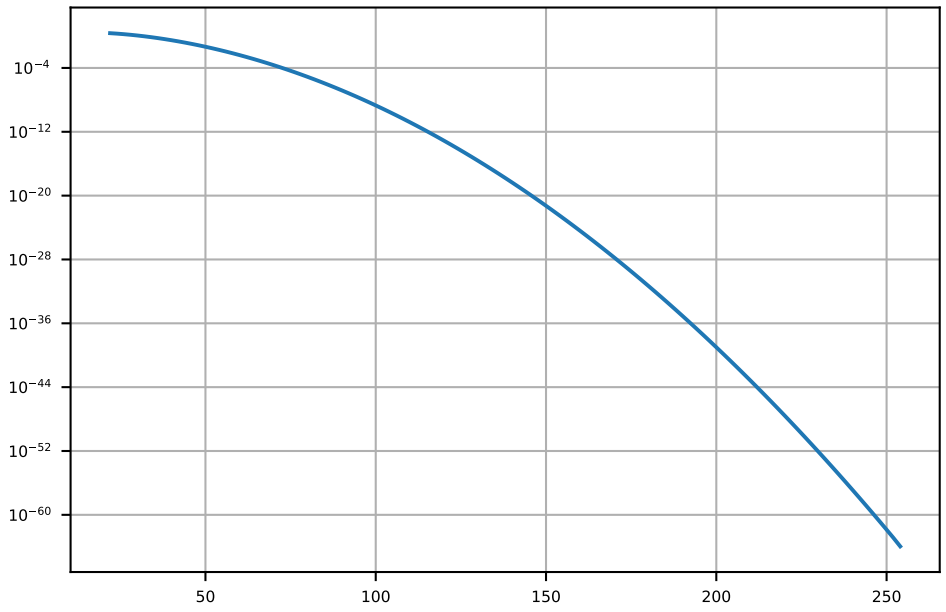
```
Out[15]: Text(0.5, 1.0, 'Ricker wavelet, w=69 a=7.3')
```



In [16]:
```
fig, ax = plt.subplots()
r = list(range(60, 120))
ax.plot(r, [sum(sig.ricker(w, a)) for w in r])
ax.set_yscale('log')
ax.grid()
```



```
In [17]: fig, ax = plt.subplots()
         sw = 256
         w = sig.ricker(sw, a)
         r = list(range(1, sw//2 - 10))
         d = [-sum(w[:i]) - sum(w[-i:]) for i in r]
         ax.plot([sw-2*x for x in r], d)
         ax.set_yscale('log')
         ax.grid()
```



2.3 Demodulation algorithm and testing function

The following function contains our prototype demodulation algorithm implementation along with test code applying it to simulated input data. By repeatedly running this function while sweeping parameters we can create plots of our algorithm's performance under various conditions.

```
In [18]: def run_ser_test(sample_duration=128, nbits=6, signal_amplitude=2.0e-3, decimation=10

             # Generate test data for this simulation run
             test_data, signal = generate_test_signal(sample_duration, nbits, signal_amplitude

             # === Begin of our prototype demodulation algorithm. ===
             # (1) Correlate the input signal against all 2^n+1 gold sequences using the corre
```

```
cor_an = correlate(signal, nbits=nbits, decimation=decimation)

# span to compute average power measurements for peak finding over, in samples
power_avg_width = int(power_avg_width * (2**nbits - 1) * decimation)

bit_period = (2**nbits) * decimation # duration of one DSSS symbol
peak_group_threshold = 0.05 * bit_period # Duration over which to group several d
hole_patching_threshold = 0.01 * bit_period # Duration over which to ignore tempo

# (2) Calculate continuous wavelet transform of correlator output and a ricker wa
# determined empirically. This transform acts like a sharpening filter on our pea
cwt_res = np.array([ sig.cwt(row, sig.ricker, [0.73 * decimation]).flatten() for
if ax:
    ax.grid()
    ax.plot(cwt_res.T)

# (3) Threshold CWT'ed correlator outputs using the factors defined above. Classi
# larger than the average of the surrounding signal.
th = np.array([ np.convolve(np.abs(row), np.ones((power_avg_width,))/power_avg_wi

# Helper function for thresholding
def compare_th(elem):
    idx, (th, val) = elem
    #print('compare_th:', th.shape, val.shape)
    return np.any(np.abs(val) > th*threshold_factor)

# (4) Group samples above threshold value into spans
peaks = [ list(group) for val, group in itertools.groupby(enumerate(zip(th.T, cwt
peaks_processed = []
peak_group = []
# For each span of samples above threshold, try to coalesce this span with adjace
for group in peaks:
    pos = np.mean([idx for idx, _val in group])
    #pol = np.mean([max(val.min(), val.max(), key=abs) for _idx, (_th, val) in gr
    pol = max([max(val.min(), val.max(), key=abs) for _idx, (_th, val) in group],
    pol_idx = np.argmax(np.bincount([ np.argmax(np.abs(val)) for _idx, (_th, val)
    peaks_processed.append((pos, pol, pol_idx))
    #print(f'group', pos, pol, pol_idx)
    #for pol, (_idx, (_th, val)) in zip([max(val.min(), val.max(), key=abs) for _
    #    print('    ', pol, val)
    #if ax:
    #    ax.axvline(pos, color='cyan', alpha=0.3)
    msg = f'peak at {pos} = {pol} idx {pol_idx}: '

    if peak_group:
        msg += f'continuing previous group: {peak_group[-1]},'
        group_start, last_pos, last_pol, peak_pos, last_pol_idx = peak_group[-1]
```

```
            if abs(pol) > abs(last_pol):
                msg += 'larger, '
                if ax:
                    ax.axvline(pos, color='magenta', alpha=0.5)
                peak_group[-1] = (group_start, pos, pol, pos, pol_idx)

            else:
                msg += 'smaller, '
                if ax:
                    ax.axvline(pos, color='blue', alpha=0.5)
                peak_group[-1] = (group_start, pos, last_pol, peak_pos, last_pol_idx)
        else:
            last_pos = None

        if not peak_group or pos - last_pos > peak_group_threshold:
            msg += 'terminating, '
            if peak_group:
                msg += f'previous group: {peak_group[-1]},'
                peak_pos = peak_group[-1][3]
                if ax:
                    ax.axvline(peak_pos, color='red', alpha=0.6)
                #ax3.text(peak_pos-20, 2.0, f'{0 if pol < 0 else 1}', horizontalalignm

            msg += f'new group: {(pos, pos, pol, pos, pol_idx)} '
            peak_group.append((pos, pos, pol, pos, pol_idx))
            if ax:
                ax.axvline(pos, color='cyan', alpha=0.5)

        if debug_range:
            low, high = debug_range
            if low < pos < high:
                print(msg)
                print(group)

    # Calculate average magnitude of all found peaks for normalization in next step
    avg_peak = np.mean(np.abs(np.array([last_pol for _1, _2, last_pol, _3, _4 in peak_
    print('avg_peak', avg_peak)

    # (5) Perform Maximum likelihood estimation to group peaks into chains of peaks s
    noprint = lambda *args, **kwargs: None
    def mle_decode(peak_groups, print=print):
        """ Maximum likelihood estimation decoding.

        This function tries to find sequences of peaks that are spaced at one-symbol
        A sequence is evaluated better the higher its peaks, the closer they match on
        # For each peak, extract index inside capture (in samples), polarity and the
        peak_groups = [ (pos, pol, idx) for _1, _2, pol, pos, idx in peak_groups ]
```

```
    # Initially populate candidate array with all peaks in first couple of symbol
    candidates = [ (abs(pol)/avg_peak, [(pos, pol, idx)]) for pos, pol, idx in pea

    # Iterate while there are candidates remaining
    while candidates:
        chain_candidates = [] # candidates for next iteration
        for chain_score, chain in candidates:
            pos, ampl, _idx = chain[-1]
            score_fun = lambda pos, npos, npol: pol_score_factor*abs(npol)/avg_pea

            # For this candidate, consider all peaks that might extend it to a lo
            next_candidates = sorted([ (score_fun(pos, npos, npol), npos, npol, ni

            print(f'    candidates for {pos}, {ampl}:')
            for score, npos, npol, nidx in next_candidates:
                print(f'        {score:.4f} {npos:.2f} {npol:.2f} {nidx:.2f}')

            nch, cor_len = cor_an.shape
            if cor_len - pos < 1.5*bit_period or not next_candidates:
                # If we have hit the end of our signal or if we did not detect an
                score = sum(score_fun(opos, npos, npol) for (opos, _opol, _oidx),
                yield score, chain

            else:
                # If we have not yet hit the end of our signal, and we still have
                # Calculate the score of the resulting extended chains and if the
                print('extending')
                for score, npos, npol, nidx in next_candidates[:3]:
                    if score > 0.5:
                        new_chain_score = chain_score * 0.9 + score * 0.1
                        chain_candidates.append((new_chain_score, chain + [(npos,

        # For next iteration select top-n highest scoring candidates just compute
        print('chain candidates:')
        for score, chain in sorted(chain_candidates, reverse=True):
            print('    ', [(score, [(f'{pos:.2f}', f'{pol:.2f}') for pos, pol, _i
        candidates = [ (chain_score, chain) for chain_score, chain in sorted(chai

# Group peaks into chains and select the chain with the highest score
res = sorted(mle_decode(peak_group, print=noprint), reverse=True)
#for i, (score, chain) in enumerate(res):
#    print(f'Chain {i}@{score:.4f}: {chain}')
(_score, chain), *_ = res

def viz(chain, peaks):
    last_pos = None
    for pos, pol, nidx in chain:
        if last_pos:
```

```
                delta = int(round((pos - last_pos) / bit_period))
                if delta > 1:
                    print(f'skipped {delta-1} symbols at {pos}/{last_pos}')

                    # Hole patching routine
                    for i in range(1, delta):
                        est_pos = last_pos + (pos - last_pos) / delta * i

                        icandidates = [ (ipos, ipol, iidx) for ipos, ipol, iidx in pea
                        if not icandidates:
                            yield None
                            continue

                        ipos, ipol, iidx = max(icandidates, key = lambda e: abs(e[1]))

                        decoded = iidx*2 + (0 if ipol < 0 else 1)
                        print(f'interpolating, last_pos={last_pos}, delta={delta}, pos
                        yield decoded

            decoded = nidx*2 + (0 if pol < 0 else 1)
            yield decoded
            if ax:
                ax.axvline(pos, color='blue', alpha=0.5)
                ax.text(pos-20, 0.0, f'{decoded}', horizontalalignment='right', vertic

            last_pos = pos

    decoded = list(viz(chain, peaks_processed))
    print('decoding [ref|dec]:')
    match_result = []
    for shift in range(-ser_maxshift, ser_maxshift):
        msg = f'=== shift = {shift} ===\n'
        failures = -shift if shift < 0 else 0 # we're skipping the first $shift symbo
        a = test_data if shift > 0 else test_data[-shift:]
        b = decoded if shift < 0 else decoded[shift:]
        for i, (ref, found) in enumerate(itertools.zip_longest(a, b)):
            if ref is None: # end of signal
                break
            msg += f'{ref if ref is not None else -1:>3d}|{found if found is not None
            if ref != found:
                failures += 1
            if i%8 == 7:
                msg += '\n'
        match_result.append((failures, msg))
    failures, msg = min(match_result, key=lambda e: e[0])
    print(msg)
    ser = failures/len(test_data)
    print(f'Symbol error rate e={ser}: {failures}/{len(test_data)}')
```

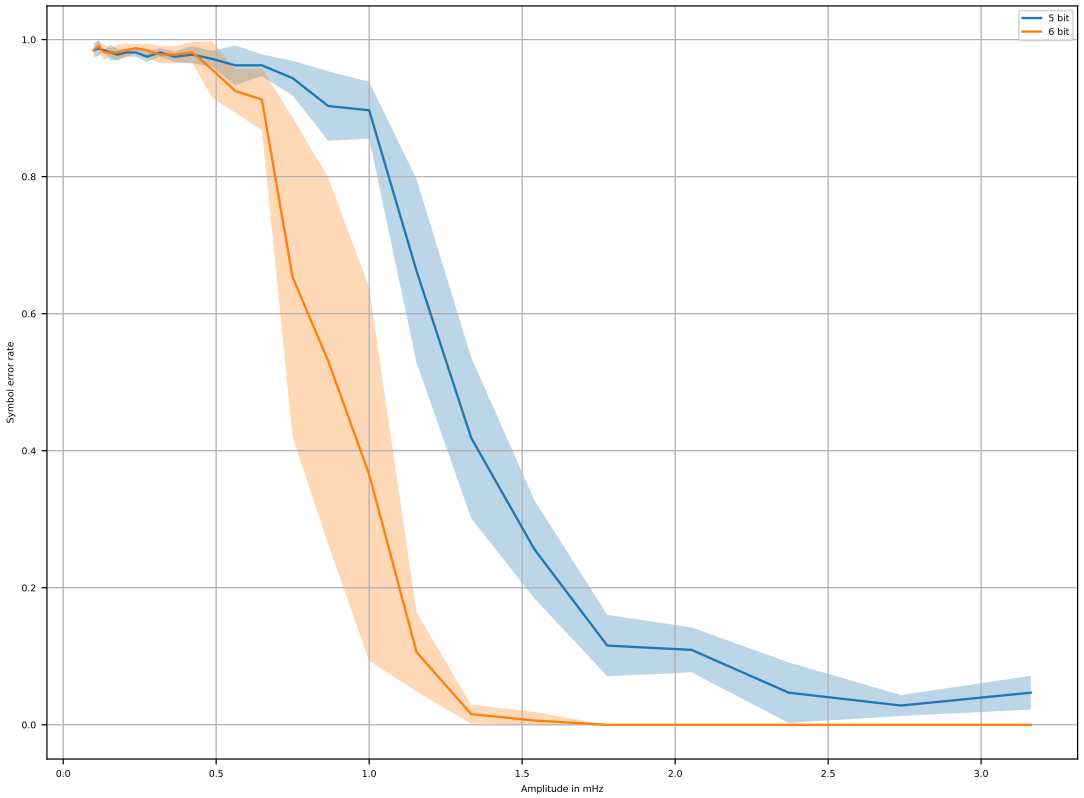
```
            br = sampling_rate / decimation / (2**nbits) * nbits * (1 - ser) * 3600
            print(f'maximum bitrate r={br} b/h')
            return ser, br

In [19]: default_params = dict(
                decimation=10,
                power_avg_width=2.5,
                max_lookahead=6.5)

        fig, ax = plt.subplots(figsize=(12, 9))

        def calculate_ser(v, seed, nbits, thf, reps, duration):
            st = np.random.RandomState(seed)
            params = dict(default_params)
            params['signal_amplitude'] = v
            params['nbits'] = nbits
            params['threshold_factor'] = thf
            sers, brs = [], []
            for i in range(reps):
                seed = st.randint(0xffffffff)
                try:
                    ser, br = run_ser_test(**params, sample_duration=duration, print=noprint,
                    sers.append(ser)
                    brs.append(br)
                except Exception as e:
                    traceback.print_exc()
                    print('got', e, 'seed', seed, 'params', params)
                    #sers.append(1.0)
                    #brs.append(0.0)
                #print(f'nbits={nbits} ampl={v:>.5f} seed={seed:08x} > ser={ser:.5f}')
            sers, brs = np.array(sers), np.array(brs)
            ser, std = np.mean(sers), np.std(sers)
            #print(f'signal_amplitude={v:<.5f}: ser={ser:<.5f} ś{std:<.5f}, br={np.mean(brs):
            return ser, std

        results = {}
        with tqdm(total = 0) as tq:
            with multiprocessing.Pool(multiprocessing.cpu_count()//2) as pool:
                for nbits, thf, reps, points, duration in [(5, 4.0, 5, 25, 64), (6, 4.0, 5, 2
                    #print(f'nbits={nbits}')
                    st = np.random.RandomState(0)
                    vs = 0.1e-3 * 10 ** np.linspace(0, 1.5, points)
                    results[nbits] = [ pool.apply_async(calculate_ser, (v, st.randint(0xfffff
                    tq.total += len(vs)
                    tq.refresh()

                pool.close()
                pool.join()
```

```
                print(f'scheduled {tq.total} tasks. waiting...')
                results = { nbits: [ res.get() for res in series ] for nbits, series in resul
                print('done')

        with open(f'dsss_experiments_res-{datetime.datetime.now():%Y-%m-%d %H:%M:%S}.json', '
            json.dump(results, f)

        for nbits, res in results.items():
            data = np.array(res)
            sers, stds = data[:,0], data[:,1]

            l, = ax.plot(vs*1e3, np.clip(sers, 0, 1), label=f'{nbits} bit')
            ax.fill_between(vs*1e3, np.clip(sers + stds, 0, 1), np.clip(sers - stds, 0, 1), fa
        ax.grid()
        ax.set_xlabel('Amplitude in mHz')
        ax.set_ylabel('Symbol error rate')
        ax.legend()
```

```
got not enough values to unpack (expected at least 1, got 0) seed 2919184728 params {'decimati
```

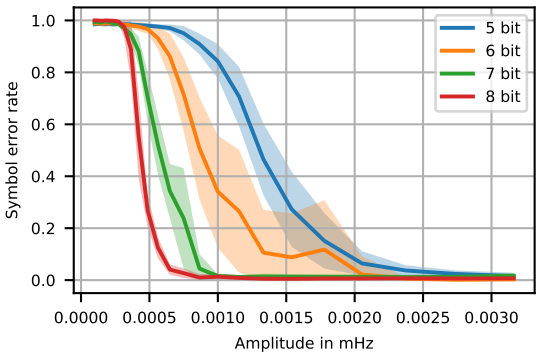
```
got not enough values to unpack (expected at least 1, got 0) seed 2642033202 params {'decimati
scheduled 50 tasks. waiting...
done
```

```
Out[19]: <matplotlib.legend.Legend at 0x7f5bd3fb75b0>
```



In [30]:
```
fig, ax = plt.subplots(figsize=(3, 2))

#     sers, brs = np.array(sers), np.array(brs)
#     ser, std = np.mean(sers), np.std(sers)
#     results = { nbits: [ res.get() for res in series ] for nbits, series in results.

with open(f'data/dsss_experiments_res-2020-02-19-19-30-05.json', 'r') as f:
    results = json.load(f)

for nbits, series in results.items():
    series = [ [ mean for mean, _std, _msg in reps if mean is not None ] for reps in
    sers = np.array([ np.mean(values) for values in series ])
    stds = np.array([ np.std(values)  for values in series ])

    # FIXME HACK HACK HACK
    vs = 0.1e-3 * 10 ** np.linspace(0, 1.5, 25)

    l, = ax.plot(vs, np.clip(sers, 0, 1), label=f'{nbits} bit')
    ax.fill_between(vs, np.clip(sers + stds, 0, 1), np.clip(sers - stds, 0, 1), facec
ax.grid()
ax.set_xlabel('Amplitude in mHz')
```

```
        ax.set_ylabel('Symbol error rate')
        ax.legend()
        fig.tight_layout()
        fig.savefig('fig_out/dsss_gold_nbits_overview.pdf')
```



In [29]:
```
default_files = [
    #    'data/dsss_experiments_res-2020-02-20-12-18-35.json',
    #    'data/dsss_experiments_res-2020-02-20-12-26-07.json',
    #    'data/dsss_experiments_res-2020-02-20-12-29-02.json'
        'data/dsss_experiments_res-par107-run115-0-2020-04-07-11-41-37.json',
        'data/dsss_experiments_res-par107-run115-1-2020-04-07-13-23-42.json',
        'data/dsss_experiments_res-par107-run115-2-2020-04-07-08-57-38.json',
        'data/dsss_experiments_res-par107-run115-3-2020-04-07-15-48-04.json',
    ]

def load_results(*files):
    results = []
    for fn in files:
        with open(fn, 'r') as f:
            results += json.load(f)
    return results

def thf_dependence_plot(results, plot_nbits=6,
                        ax=None, cbar_ax=None, intercept_ax=None,
                        xlabel=True, x2label=False, ylabel=True, y2label=True, y2ticks
                        legend_loc=4, split_legend=False, calc_best_ampl=False):

    thfs = [thf for (_nbits, thf, _reps, _points, _duration, _decimation), series in
    cmap = matplotlib.cm.viridis
    cm_func = lambda x: cmap((x - min(thfs)) / (max(thfs) - min(thfs)))

    thf_sers = {}
    for (nbits, thf, reps, points, duration, decimation), series in results:
```

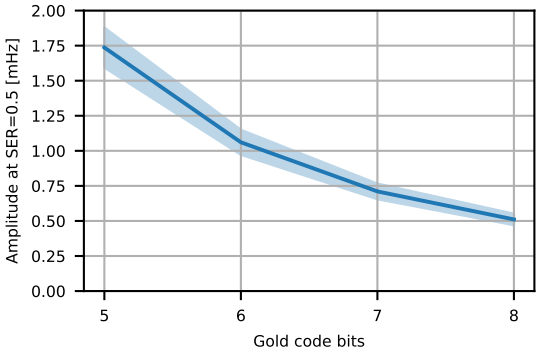
```
        if nbits != plot_nbits:
            continue
        data = [ [ mean for mean, _std, _msg in reps if mean is not None ] for _amp, 
        amps = [ amp*1000 for amp, _reps in series ]
        sers = np.array([ np.mean(values) for values in data ])
        stds = np.array([ np.std(values)  for values in data ])
        thf_sers[thf] = list(zip(amps, sers, stds))

        if ax:
            l, = ax.plot(amps, np.clip(sers, 0, 1), label=f'thf={thf}', color=cm_func
            ax.fill_between(amps, np.clip(sers + stds, 0, 1), np.clip(sers - stds, 0,
            ax.axhline(0.5, color='gray', ls=(0, (3, 4)), lw=0.8)
    if ax:
        ax.grid()
        ax.set_title(f'{plot_nbits}-bit Gold code')
        if xlabel:
            ax.set_xlabel('Amplitude [mHz]')
        if ylabel:
            ax.set_ylabel('Symbol Error Rate')

    def plot_base_amp(ax):
        base_sers = {}
        for thf, sers in thf_sers.items():
            base = np.mean([ser for amp, ser, std in sorted(sers)[-2:]])
            base_std = np.sqrt(np.mean([std**2 for amp, ser, std in sorted(sers)[-2:]
            base_sers[thf] = (base, base_std)

        x = sorted(base_sers.keys())
        y = np.array([ base_sers[thf][0] for thf in x ])
        std = np.array([ base_sers[thf][1] for thf in x ])
        l = ax.plot(x, y, label='SER at large amplitudes')
        ax.fill_between(x, y-std, y+std, color=l[0].get_color(), alpha=0.3)
        return l

    def plot_intercepts(ax, SER_TH = 0.5):
        intercepts = {}
        for thf, sers in thf_sers.items():
            last_ser, last_amp, last_std = 0, 0, 0
            for amp, ser, std in sorted(sers):
                if last_ser > SER_TH and ser < SER_TH:
                    icp = last_amp + (SER_TH - last_ser) / (ser - last_ser) * (amp - 
                    ic_std = abs(last_amp - amp) / 2# np.sqrt(np.mean(last_std**2 + s
                    intercepts[thf] = (icp, ic_std)
                    break
                last_amp, last_ser = amp, ser
            else:
                intercepts[thf] = None, None
```

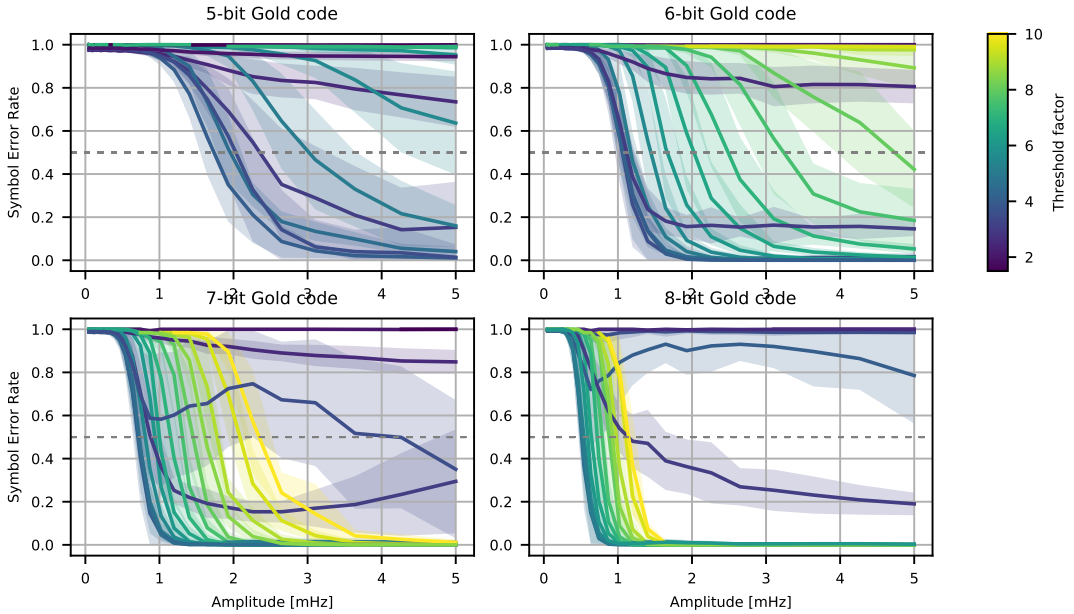
```
        ser_valid = [thf for thf, (ser, _std) in intercepts.items() if ser is not None]
        #ax.axvline(min(ser_valid), color='red')
        #ax.axvline(max(ser_valid), color='red')

        x = sorted(intercepts.keys())
        data = np.array([ intercepts[thf] for thf in x ])
        y = data[:,0]
        std = data[:,1]

        if ax:
            ax.set_xlim([min(x), max(x)])
            l = ax.plot(x, y, label='Amplitude at SER=0.5', color='orange')
        else:
            l = None

        x, y, std = zip(*[ (le_x, le_y, le_std) for le_x, le_y, le_std in zip(x, y, st
        y, std = np.array(y), np.array(std)
        if ax:
            ax.fill_between(x, y-std, y+std, color=l[0].get_color(), alpha=0.3)

            trans = matplotlib.transforms.blended_transform_factory(ax.transData, ax.t
            ax.fill_between([-1, min(ser_valid)], 0, 1, facecolor='red', alpha=0.2, tr
            ax.fill_between([max(ser_valid), max(ser_valid)*10], 0, 1, facecolor='red
            ax.set_ylim([min(y)*0.9, max(y)*1.1])
            ax.grid()

        best_ampl = (np.inf, np.nan)
        for yval, stdval in zip(y, std):
            if yval < best_ampl[0]:
                best_ampl = [yval, stdval]

        return l, best_ampl

    if intercept_ax:
        if isinstance(intercept_ax, tuple):
            intercept_ax, intercept_ax_twin = intercept_ax
        else:
            intercept_ax_twin = intercept_ax.twinx()

    if intercept_ax or calc_best_ampl:
        l1, best_ampl = plot_intercepts(intercept_ax)
    else:
        best_ampl = None

    if intercept_ax:
        l2 = plot_base_amp(intercept_ax_twin)

        intercept_ax.set_title(f'{plot_nbits}-bit Gold code')
```

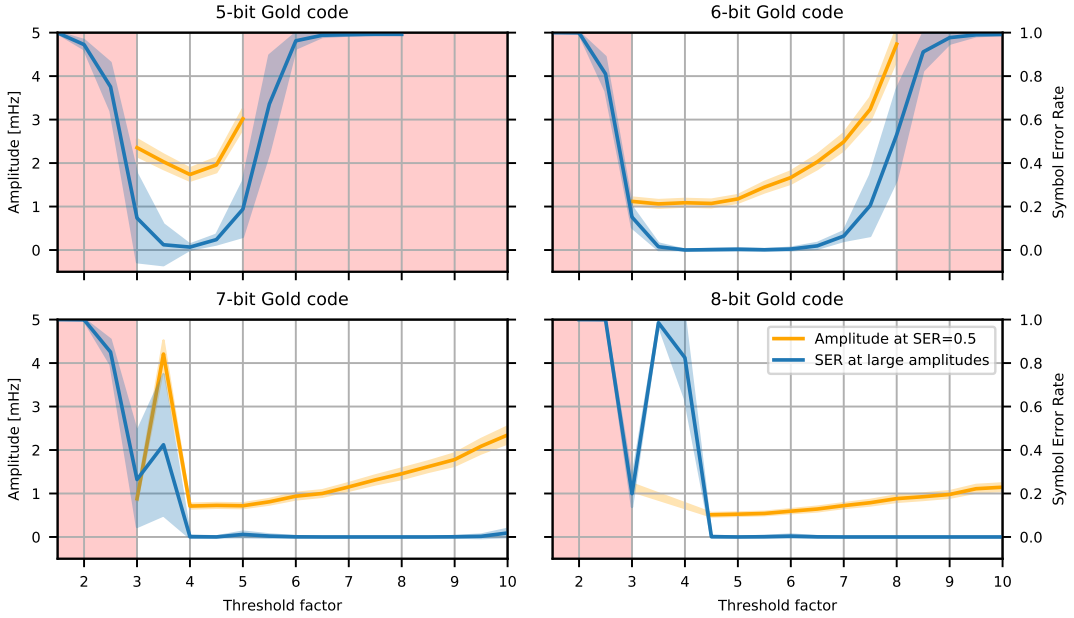
```
            if xlabel:
                intercept_ax.set_xlabel('Threshold factor')
            if x2label:
                intercept_ax_twin.set_xlabel('Threshold factor')
            if ylabel:
                intercept_ax.set_ylabel('Amplitude [mHz]')
            intercept_ax.set_ylim(ic_ylim)
            intercept_ax_twin.set_ylim([-0.1, 1])
            if y2label:
                intercept_ax_twin.set_ylabel('Symbol Error Rate')
            if not y2ticks:
                intercept_ax_twin.set_yticklabels([])
            if legend_loc is not None:
                if split_legend:
                    intercept_ax.legend(l1, [l1[0].get_label()], loc=legend_loc)
                    intercept_ax_twin.legend(l2, [l2[0].get_label()], loc=legend_loc)
                else:
                    intercept_ax.legend(l1 + l2, [l.get_label() for l in l1+l2], loc=leger

        if cbar_ax:
            norm = matplotlib.colors.Normalize(vmin=min(thfs), vmax=max(thfs))
            cb1 = matplotlib.colorbar.ColorbarBase(cbar_ax, cmap=cmap, norm=norm, orienta

        return best_ampl

    import warnings
    warnings.filterwarnings('ignore')


    def plot_gold_sensitivity(results, nbitses=[5,6,7,8]):
        nbitses = np.array(nbitses)
        ampls = np.array([ thf_dependence_plot(plot_nbits=dep_n, results=results, calc_bes
        fig, ax = plt.subplots(figsize=(3, 2))
        l = ax.plot(nbitses, ampls[:,0])
        ax.fill_between(nbitses, ampls[:,0]-ampls[:,1], ampls[:,0]+ampls[:,1], color=l[0]
        ax.grid()
        ax.set_xlabel('Gold code bits')
        ax.set_ylabel('Amplitude at SER=0.5 [mHz]')
        ax.set_ylim([0, 2])
        ax.xaxis.set_major_locator(ticker.MultipleLocator(1.0))
        fig.tight_layout()
        return fig

    def plot_amplitude_ber(results, grid=(2, 3), nbitses=[5,6,7,8], figsize=(12, 9), xlim=
        fig = plt.figure(figsize=figsize)
        gs = plt.GridSpec(*grid, figure=fig, width_ratios=[1, 1, 0.05])

        cbar_ax = fig.add_subplot(gs[0, 2])
```

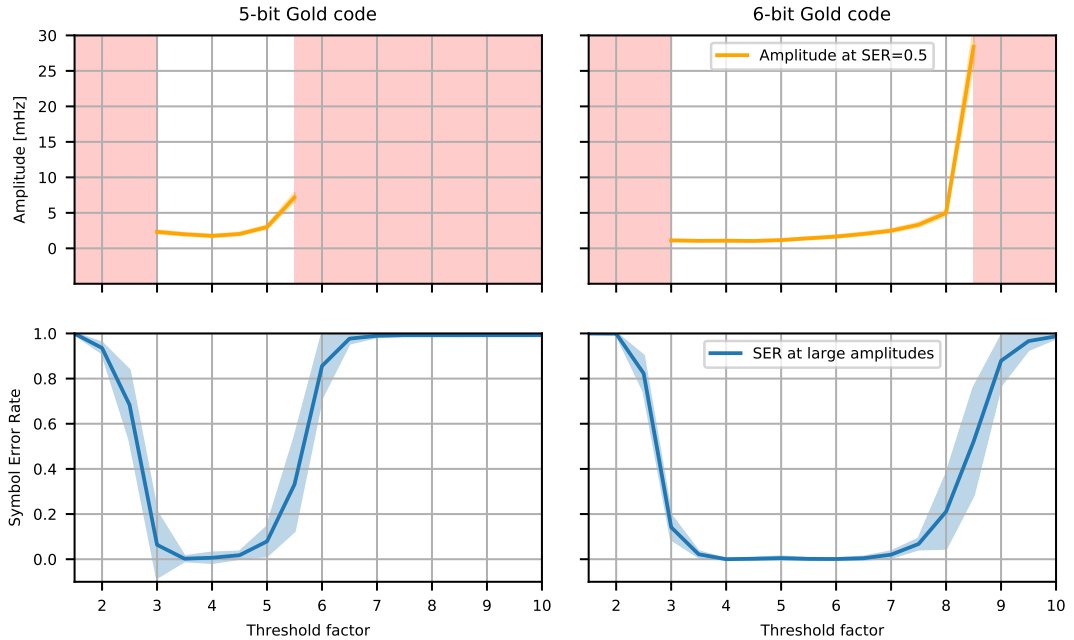
```
    axs = np.empty([2, 2], dtype=object)
    for i, nbits in enumerate(nbitses):
        row, col = i//2, i%2

        ax = axs[row, col] = fig.add_subplot(gs[row, col])
        if xlog:
            ax.set_xscale('log')
        if xlim is not None:
            ax.set_xlim(xlim)
        if row == 1:
            ax.get_shared_x_axes().join(axs[0, col])
        if col == 1:
            ax.get_shared_y_axes().join(axs[row, 0])

        xlabel = row==1 if len(nbitses) > 2 else True
        thf_dependence_plot(plot_nbits=nbits, ax=ax, cbar_ax=cbar_ax if i==0 else Non

    return fig

def plot_thf_graph(results, nbitses=[5,6,7,8], ic_ylim=[-0.5, 5], figsize=(12, 9)):
    fig, axs = plt.subplots(2, 2, figsize=figsize, sharex='col', sharey='row', gridsp
    for nbits, ax, ax_below in zip(nbitses, axs.flatten(), [*axs.flatten()[2:], None,
        if len(nbitses) <= 2:
            ax = ax, ax_below
            ax_below.grid()
            y2label = nbits in [5, 7]
            legend_loc = 9
            y2ticks = True
        else:
            y2ticks = y2label = nbits in [6, 8]
            legend_loc = 1
        thf_dependence_plot(plot_nbits=nbits, intercept_ax=ax,
                            xlabel=nbits in [7, 8], ylabel=nbits in [5, 7], y2label=y
                            y2ticks=y2ticks, x2label=len(nbitses) <= 2,
                            legend_loc=legend_loc if nbits == nbitses[-1] else None,
                            split_legend =  len(nbitses) <= 2,
                            results=results)
    return fig

plot_gold_sensitivity(load_results(*default_files))\
.savefig('fig_out/dsss_gold_nbits_sensitivity.pdf');

plot_amplitude_ber(load_results(*default_files), figsize=(7, 4))\
.savefig('fig_out/dsss_thf_amplitude_5678.pdf');

plot_thf_graph(load_results(*default_files), figsize=(7, 4))\
.savefig('fig_out/dsss_thf_sensitivity_5678.pdf')
```

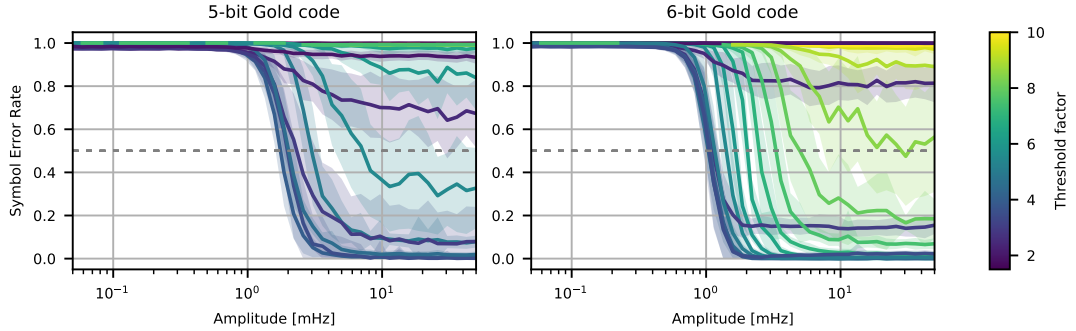
```
# Note: due to a mistake these "par114" files actually contain "par115" data.
plot_thf_graph(load_results('data/dsss_experiments_res-par114-run120-0-2020-04-08-20-0
               nbitses=[5,6], ic_ylim=[-4.99, 30],
              figsize=(7, 4))\
.savefig('fig_out/dsss_thf_sensitivity_56.pdf');

plot_amplitude_ber(load_results('data/dsss_experiments_res-par114-run120-0-2020-04-08
                  xlog=True, xlim=[5e-2, 5e1],
                  figsize=(7, 4))\
.savefig('fig_out/dsss_thf_amplitude_56.pdf');
```





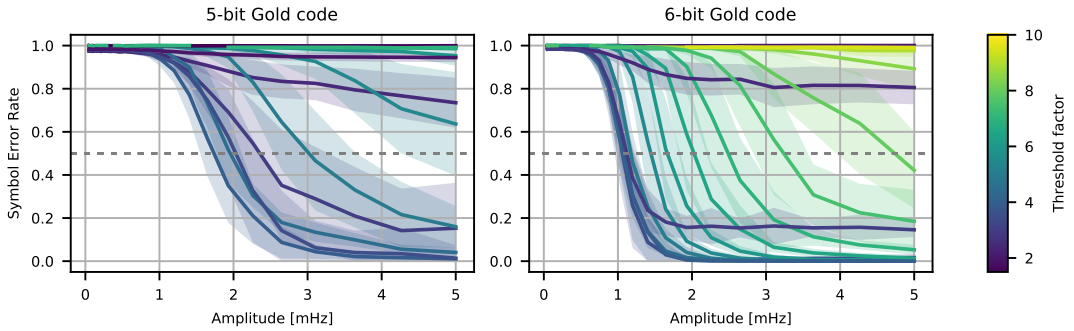

5-bit Gold code
6-bit Gold code
7-bit Gold code
8-bit Gold code
Amplitude [mHz]
Symbol Error Rate
Threshold factor
Amplitude at SER=0.5
SER at large amplitudes

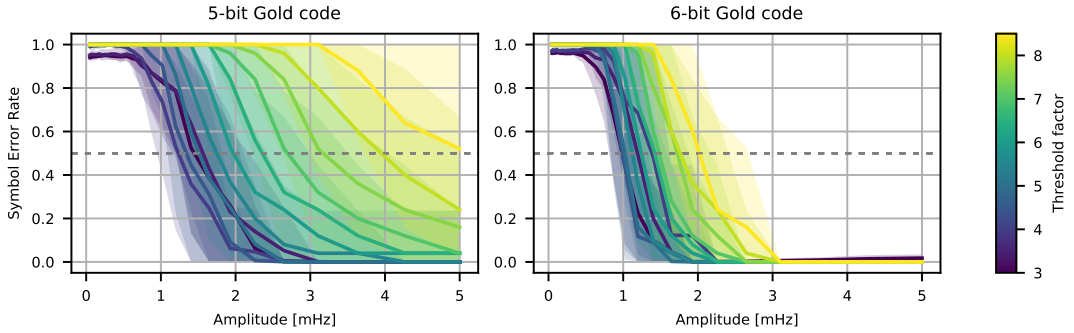

5-bit Gold code
6-bit Gold code
Amplitude [mHz]
Symbol Error Rate
Threshold factor
Amplitude at SER=0.5
SER at large amplitudes

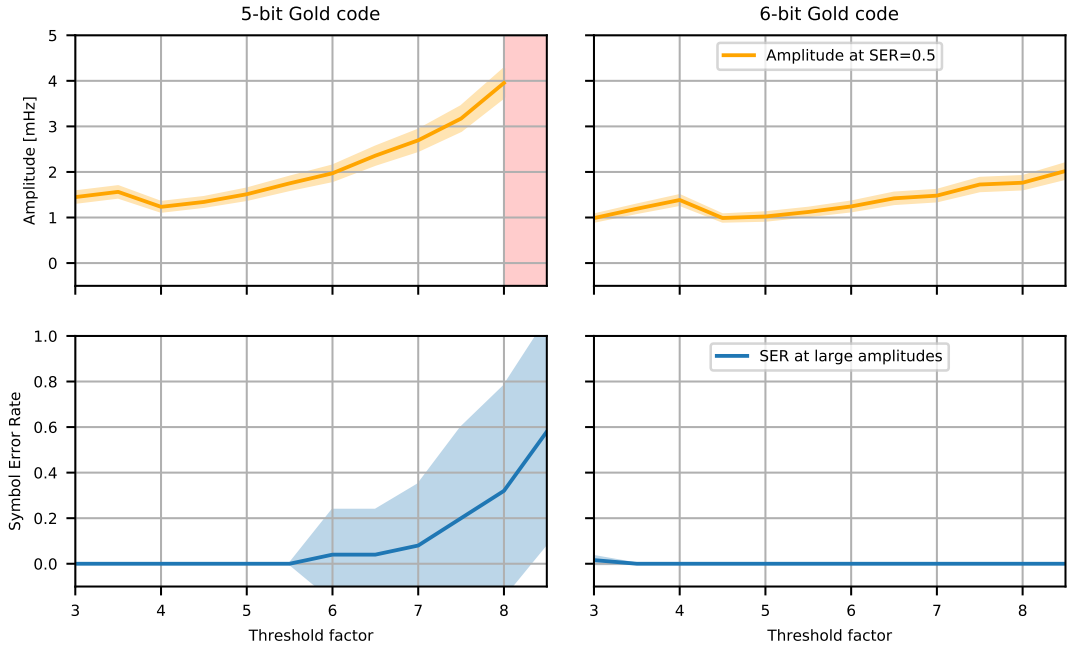



```
In [26]: def load_results_fw_sim(*files, background=None, filter_decimation=None):
             results = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: [])))

             for fn in files:
                 with open(fn, 'r') as f:
                     for (nbits, thf, decimation, symbols, seed, amp, background), result in j
                         if filter_decimation is None or decimation == filter_decimation:
                             results[background][(nbits, thf, symbols, decimation)][amp].appen

             if len(results) > 1:
                 if background is None:
                     raise ValueError('Results series contains series for multiple noise backg

                 results = results[background]
             else:
                 results = list(results.values())[0]

             out = []
             for (nbits, thf, duration, decimation), series in results.items():
                 out_series = []
                 for amplitude, amplitude_series in sorted(series.items(), key=lambda x: x[0])
                     reps = len(amplitude_series)
                     out_amplitude_series = [(ser if ser is not None else 1.0, None, None) for
                     out_series.append((amplitude, out_amplitude_series))
                 out.append(((nbits, thf, reps, len(series), duration, decimation), out_series
             return out

         plot_amplitude_ber(load_results(*default_files), nbitses=[5, 6], figsize=(7, 4))\
         .savefig('fig_out/dsss_thf_amplitude_56_jupyter_impl.pdf');

         fw_sim_res = load_results_fw_sim(*glob.glob('data/fw_sim_ser_2/*.json'), filter_decima
         plot_amplitude_ber(results=fw_sim_res, nbitses=[5, 6], figsize=(7, 4))\
         .savefig('fig_out/dsss_thf_amplitude_56_fw_impl.pdf');
         plot_thf_graph(results=fw_sim_res, nbitses=[5, 6], figsize=(7, 4))\
```

```
.savefig('fig_out/dsss_thf_sensitivity_56_fw_impl.pdf');
```

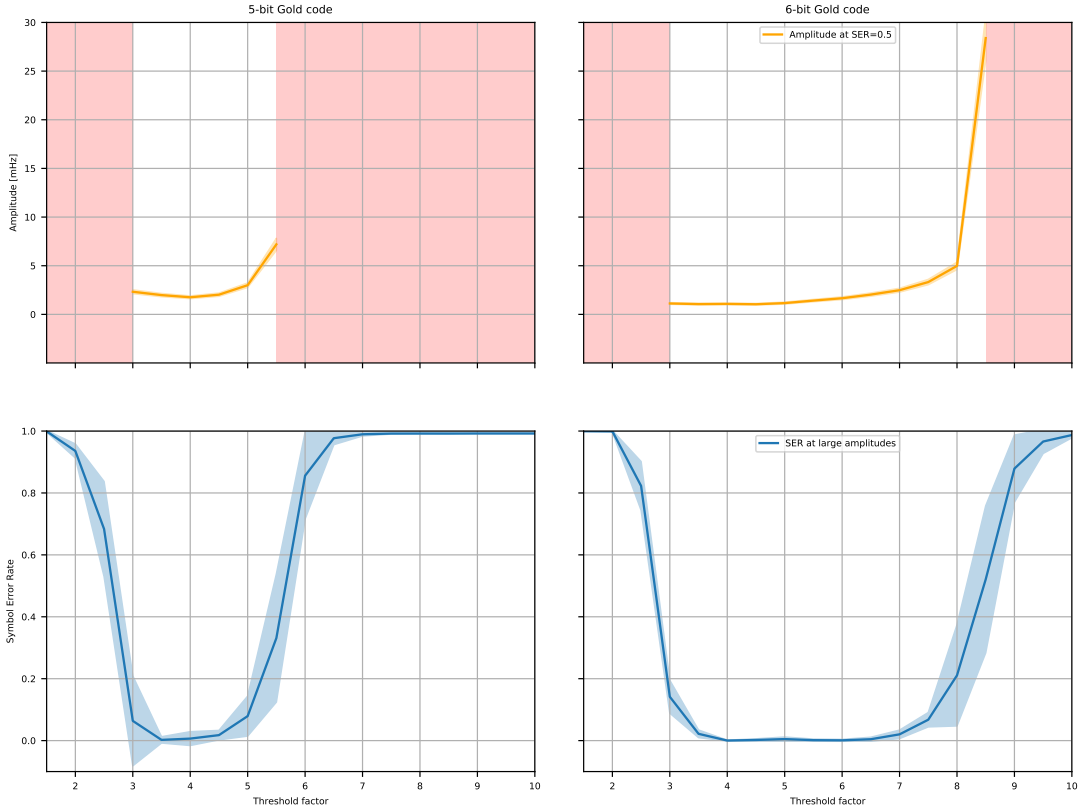


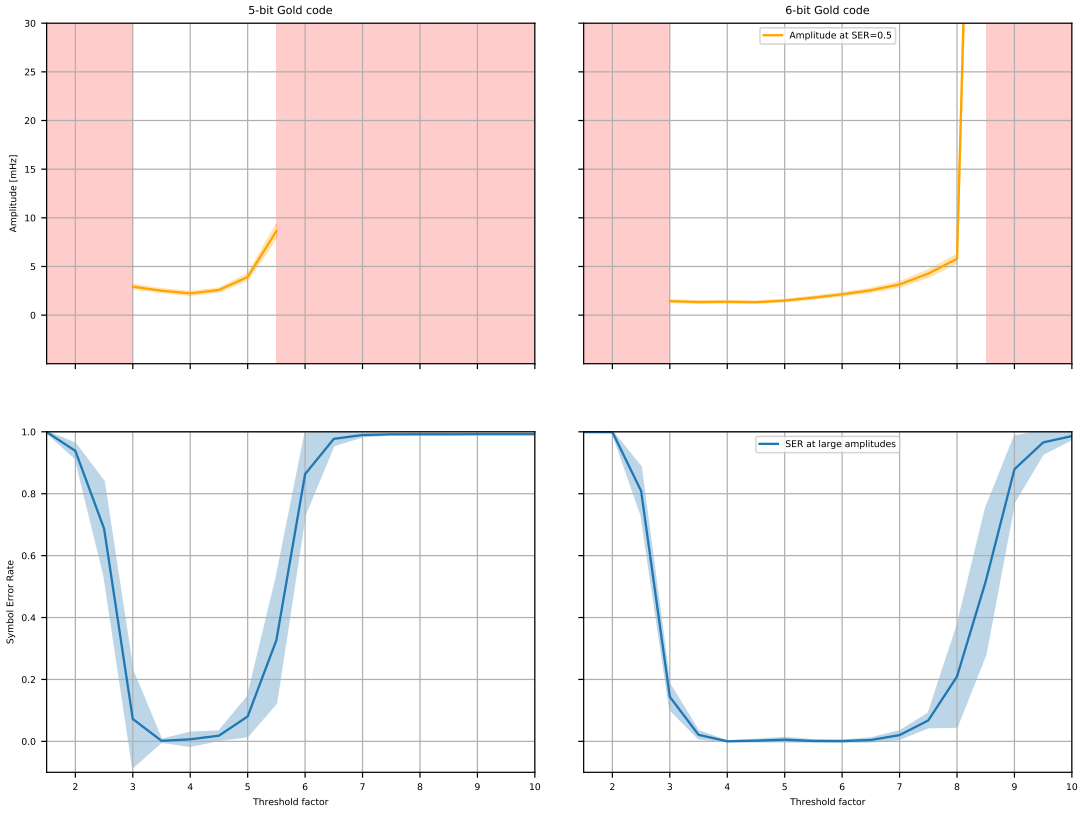


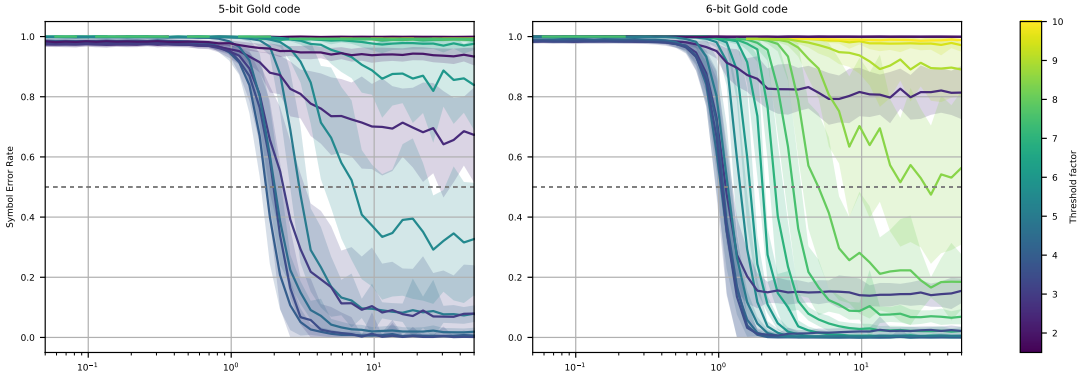


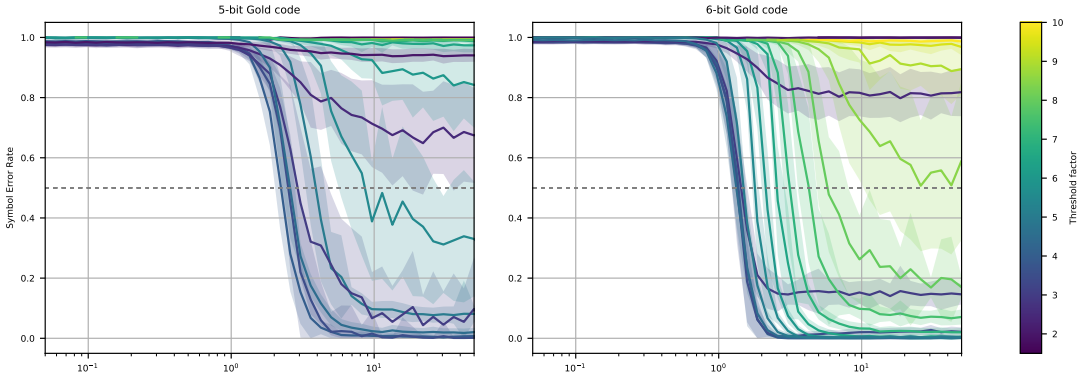
In [70]:
```
#sorted([x[0] for x in fw_sim_res])
#sorted({amp for _params, series in fw_sim_res for amp, reps in series}),\
#sorted({amp for _params, series in load_results(*default_files) for amp, reps in ser
```

In [43]:
```
extra_amp_files = ['data/dsss_experiments_res-par114-run120-0-2020-04-08-20-03-56.jso
synth_files = [
    'data/dsss_experiments_res-par115-synth-run122-0-2020-04-11-20-07-33.json',
    'data/dsss_experiments_res-par115-synth-run122-1-2020-04-11-20-39-19.json'
]

# Note: due to a mistake these "par114" files actually contain "par115" data.
plot_thf_graph(extra_amp_files, nbitses=[5,6], ic_ylim=[-4.99, 30],);
plot_thf_graph(synth_files, nbitses=[5,6], ic_ylim=[-4.99, 30]);

plot_amplitude_ber(extra_amp_files, nbitses=[5,6], xlog=True, xlim=[5e-2, 5e1]);
plot_amplitude_ber(synth_files, nbitses=[5,6], xlog=True, xlim=[5e-2, 5e1]);
```




5-bit Gold code
6-bit Gold code
Amplitude at SER=0.5
Amplitude [mHz]
Symbol Error Rate
SER at large amplitudes
Threshold factor

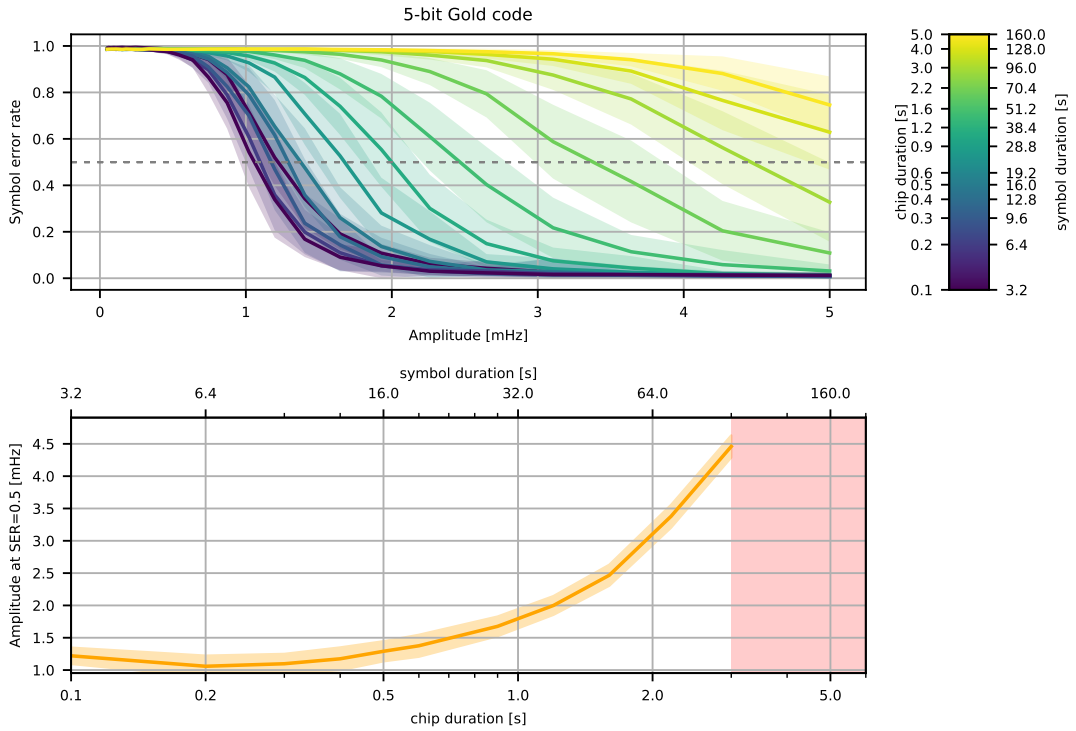

5-bit Gold code
6-bit Gold code
Symbol Error Rate
Threshold factor

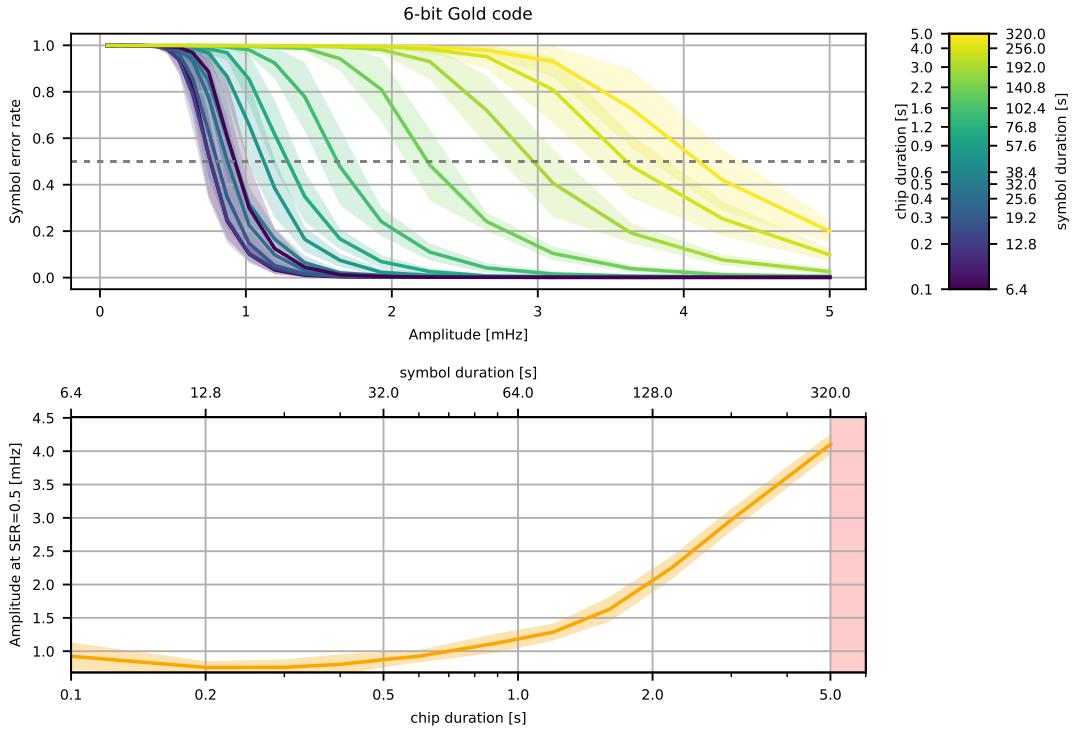


In [59]:
```
chip_duration_default_files = [
#     'data/dsss_experiments_res-2020-02-20-14-10-13.json',
#     'data/dsss_experiments_res-2020-02-20-13-21-57.json',
#     'data/dsss_experiments_res-2020-02-20-13-23-47.json',
#     'data/dsss_experiments_res-2020-02-20-19-51-21.json',
#     'data/dsss_experiments_res-2020-02-20-20-43-32.json',
#     'data/dsss_experiments_res-2020-02-20-21-36-42.json',
#     'data/dsss_experiments_res-par107-run115-0-2020-04-07-11-41-37.json',
#     'data/dsss_experiments_res-par107-run115-1-2020-04-07-13-23-42.json',
#     'data/dsss_experiments_res-par107-run115-2-2020-04-07-08-57-38.json',
#     'data/dsss_experiments_res-par107-run115-3-2020-04-07-15-48-04.json',
    'data/dsss_experiments_res-par114-run119-0-2020-04-08-20-13-44.json'
]

def plot_chip_duration_sensitivity(only_nbits=5, files=chip_duration_default_files, o
    fig, ((ax, cbar_ax), (intercept_ax, empty)) = plt.subplots(2, 2, figsize=figsize,
    empty.axis('off')
    #fig.tight_layout()

    results = []

    for fn in files:
        with open(fn, 'r') as f:
            results += json.load(f)

    decimations = [decimation for (_nbits, thf, _reps, _points, _duration, decimation
    cmap = matplotlib.cm.viridis
    cm_func = lambda x: cmap(np.log10(x - min(decimations)) / (np.log10(max(decimatio

    decimation_sers = {}
    for (nbits, thf, reps, points, duration, decimation), series in results:
        if only_thf is not None and thf != only_thf:
            continue
```

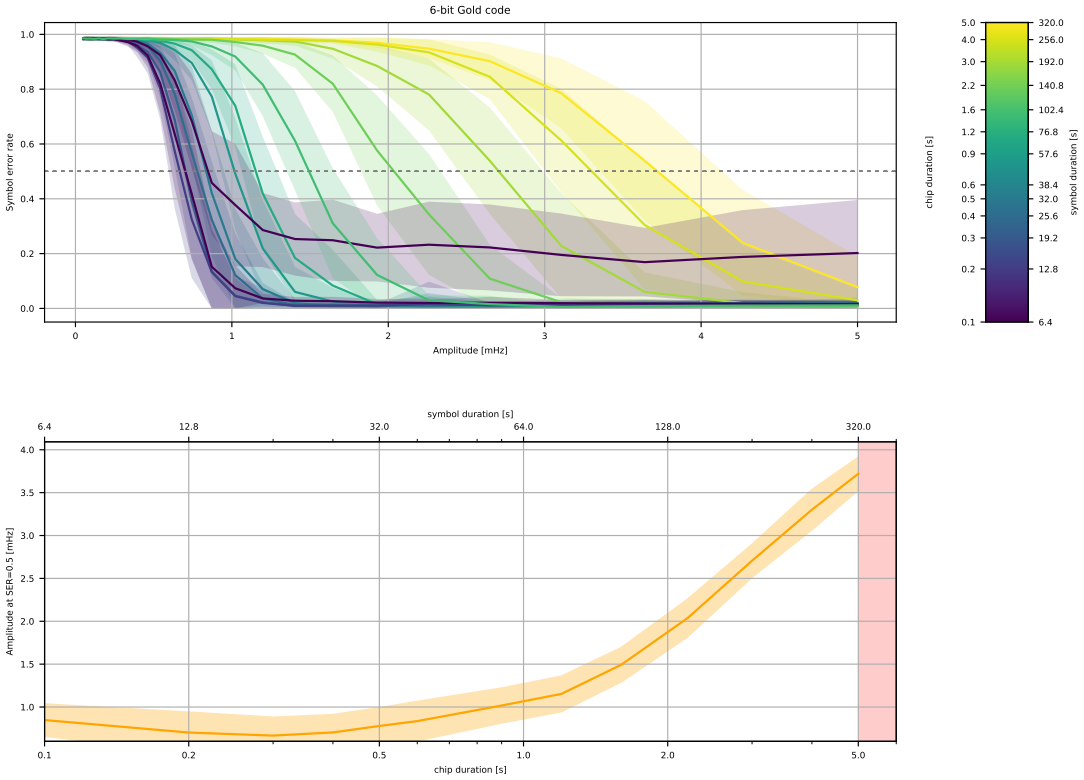
```
        if nbits != only_nbits:
            continue
        if not decimation > 0:
            continue
        data = [ [ mean for mean, _std, _msg in reps if mean is not None ] for _amp, 
        amps = [ amp for amp, _reps in series ]
        sers = np.array([ np.mean(values) for values in data ])
        stds = np.array([ np.std(values)  for values in data ])
        decimation_sers[decimation] = list(zip(amps, sers, stds))

        amps = [ amp*1000 for amp in amps ]
        l, = ax.plot(amps, np.clip(sers, 0, 1), label=f'decimation={decimation}', col
        ax.fill_between(amps, np.clip(sers + stds, 0, 1), np.clip(sers - stds, 0, 1),
        ax.axhline(0.5, color='gray', ls=(0, (3, 4)), lw=0.8)
ax.grid()
ax.set_xlabel('Amplitude [mHz]')
ax.set_ylabel('Symbol error rate')
ax.set_title(f'{only_nbits}-bit Gold code')

norm = matplotlib.colors.Normalize(vmin=np.log10(min(decimations)), vmax=np.log10
tick_decs = sorted(set(float(dec) for dec in decimations))
yticks = [np.log10(d) for d in tick_decs]
cb1 = matplotlib.colorbar.ColorbarBase(cbar_ax, cmap=cmap, norm=norm, orientation=
cb1t = cbar_ax.twinx()
cb1t.set_ylim(cbar_ax.get_ylim())
cb1t.set_yticks(yticks)

cbar_ax.set_yticklabels([f'{d/sampling_rate:.1f}' for d in tick_decs])
cbar_ax.set_ylabel("chip duration [s]", labelpad=-40)

cb1t.set_yticklabels([f'{d/sampling_rate * 2**only_nbits:.1f}' for d in tick_decs]
cb1t.set_ylabel("symbol duration [s]")


def plot_intercepts(ax, SER_TH = 0.5):
    intercepts = {}
    for dec, sers in decimation_sers.items():
        last_ser, last_amp, last_std = 0, 0, 0
        for amp, ser, std in sorted(sers):
            if last_ser > SER_TH and ser < SER_TH:
                icp = last_amp + (SER_TH - last_ser) / (ser - last_ser) * (amp - 
                ic_std = (abs(last_amp - amp) / 2) + np.sqrt(np.mean(last_std**2 
                intercepts[dec] = (icp, ic_std)
                break
            last_amp, last_ser = amp, ser
        else:
            intercepts[dec] = None, None
```

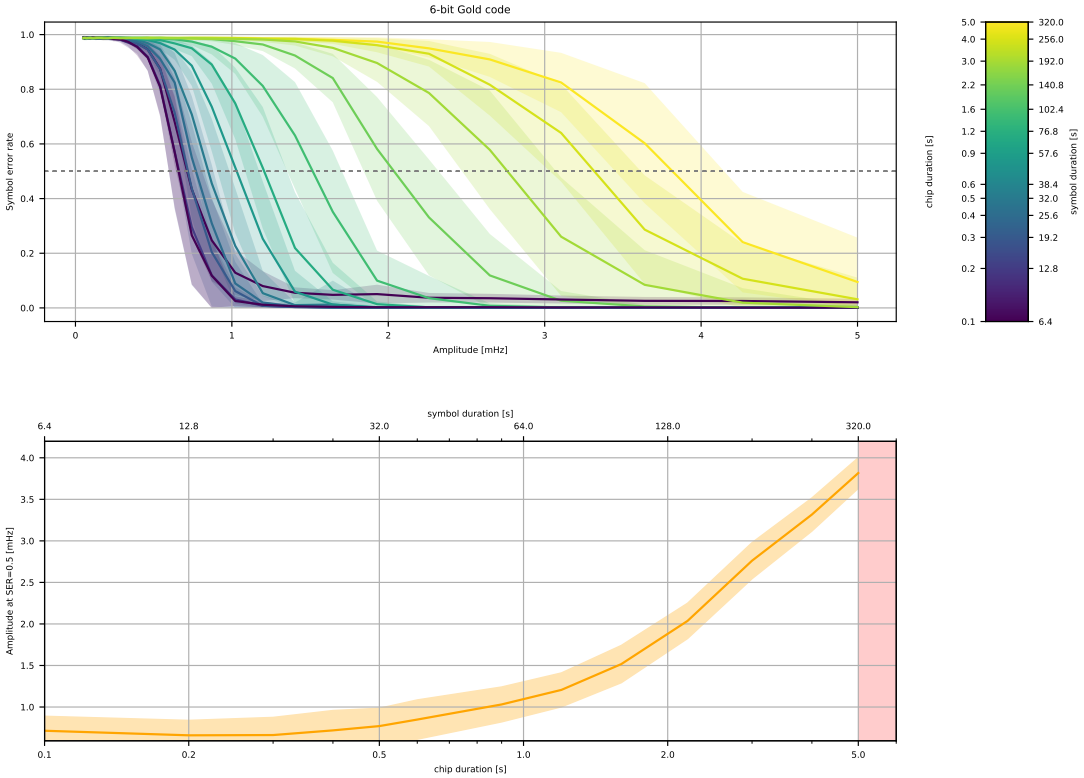
```
            ser_valid = [dec for dec, (ser, _std) in intercepts.items() if ser is not None]
            #ax.axvline(min(ser_valid), color='red')
            #ax.axvline(max(ser_valid), color='red')

            x = sorted(intercepts.keys())
            data = np.array([ intercepts[dec] for dec in x ])
            y = data[:,0]
            std = data[:,1]
            ax.set_xlim([min(x), max(x)])
            y = [ v*1000 if v is not None else v for v in y ]
            l = ax.plot(x, y, label='Amplitude at SER=0.5 [mHz]', color='orange')
            #ax.legend(loc=3)
            ax.set_ylabel('Amplitude at SER=0.5 [mHz]')
            ax.grid()

            x, y, std = zip(*[ (le_x, le_y, le_std) for le_x, le_y, le_std in zip(x, y, st
            y, std = np.array(y), np.array(std)
            ax.fill_between(x, y-std, y+std, color=l[0].get_color(), alpha=0.3)

            trans = matplotlib.transforms.blended_transform_factory(ax.transData, ax.trans
            ax.fill_between([-1, min(ser_valid)], 0, 1, facecolor='red', alpha=0.2, transf
            ax.fill_between([max(ser_valid), max(ser_valid)*10], 0, 1, facecolor='red', al
            ax.set_ylim([min(y)*0.9, max(y)*1.1])
            ax.set_xscale('log')
            ax.xaxis.set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, _: '{:g
            xticks = [1, 2, 5, 10, 20, 50]
            ax.set_xticks(xticks)
            ax.set_xticklabels([ f'{x/sampling_rate:.1f}' for x in xticks ])
            ax.set_xlim([1, 60])
            ax.set_xlabel('chip duration [s]')

            axt = ax.twiny()
            axt.set_xlim(ax.get_xlim())
            axt.set_xscale('log')
            axt.set_xticks(xticks)
            axt.set_xticklabels([ f'{x/sampling_rate * 2**only_nbits:.1f}' for x in xticks
            axt.set_xlabel('symbol duration [s]')

            return l

        l1 = plot_intercepts(intercept_ax)
        return fig

plot_chip_duration_sensitivity(5, figsize=(7, 5))\
.savefig('fig_out/chip_duration_sensitivity_5.pdf');
plot_chip_duration_sensitivity(6, figsize=(7, 5))\
.savefig('fig_out/chip_duration_sensitivity_6.pdf');
```

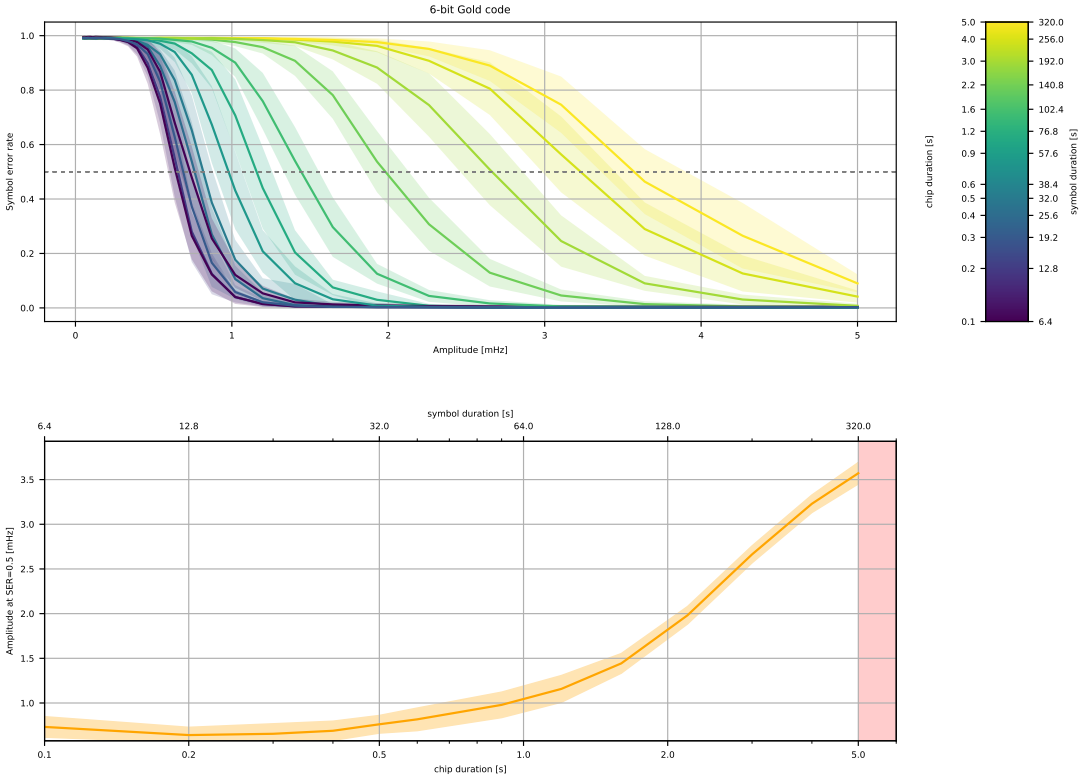

5-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]

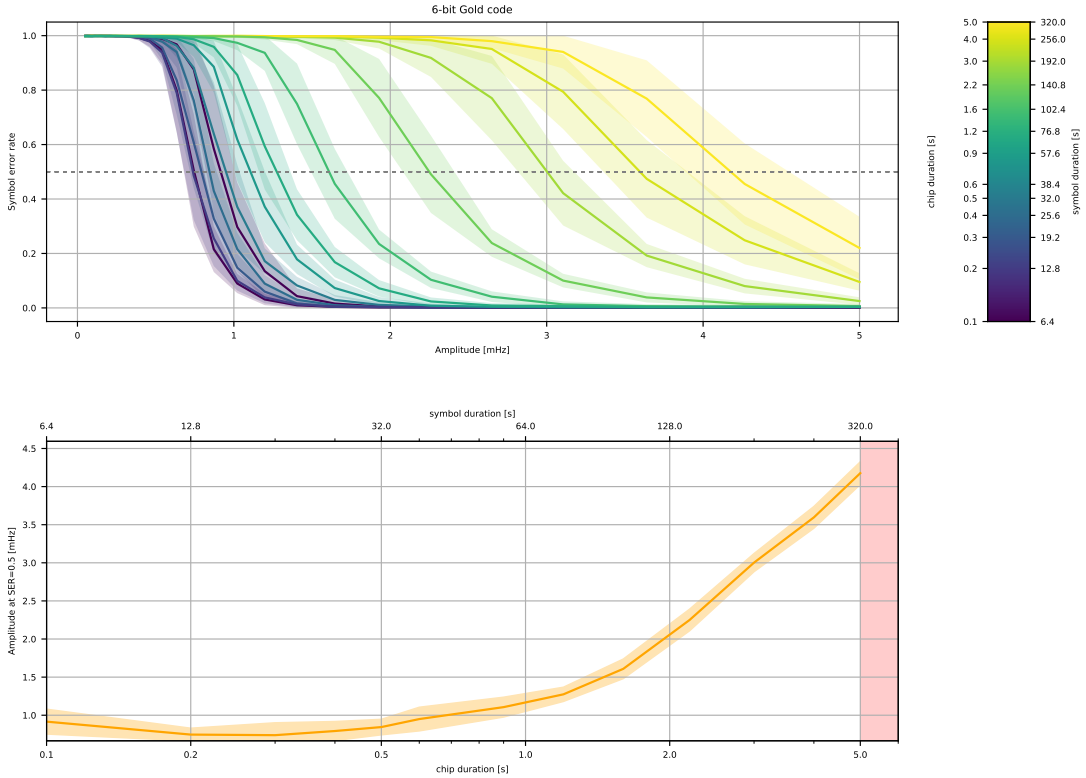

6-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]

In [45]:
```
new_files = [
        'data/dsss_experiments_res-par111-run119-0-2020-04-09-04-02-53.json',
        'data/dsss_experiments_res-par111-run119-1-2020-04-08-16-11-20.json',
        'data/dsss_experiments_res-par111-run119-2-2020-04-08-18-07-22.json',
        'data/dsss_experiments_res-par111-run119-3-2020-04-08-13-56-03.json',
              ]
plot_chip_duration_sensitivity(6, only_thf=3.5, files=new_files);
plot_chip_duration_sensitivity(6, only_thf=4.0, files=new_files);
plot_chip_duration_sensitivity(6, only_thf=4.5, files=new_files);
plot_chip_duration_sensitivity(6, only_thf=5.0, files=new_files);
#plot_chip_duration_sensitivity(6, files=new_files);
```

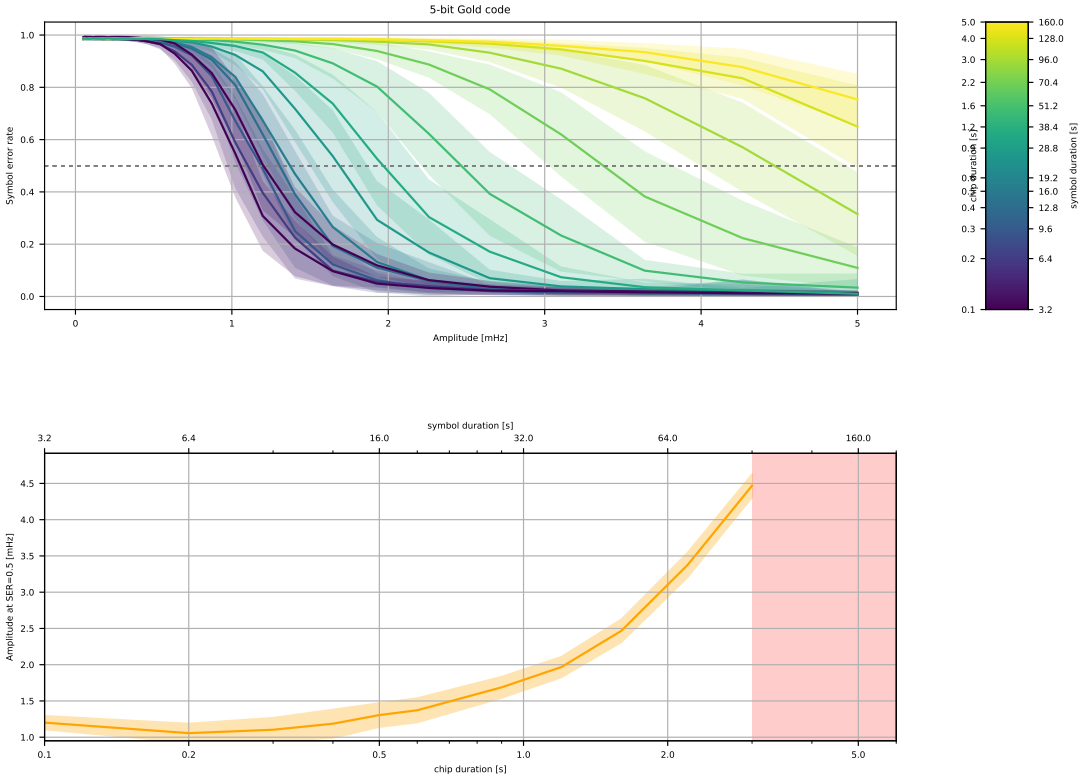


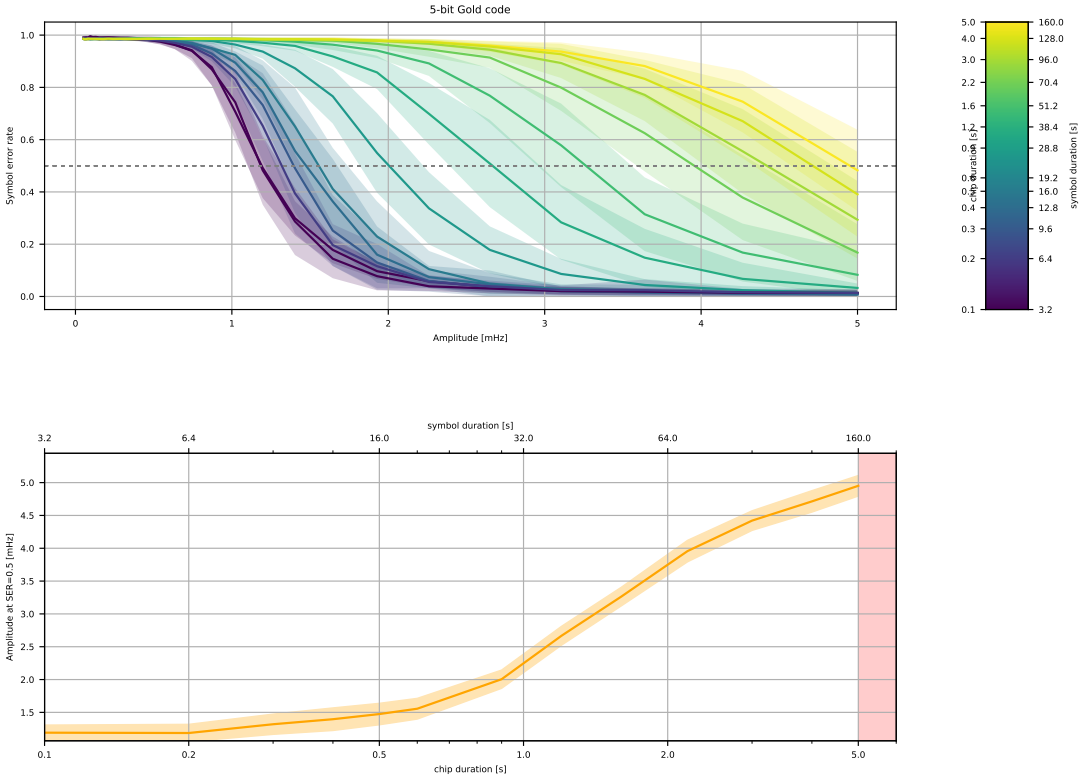

6-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]

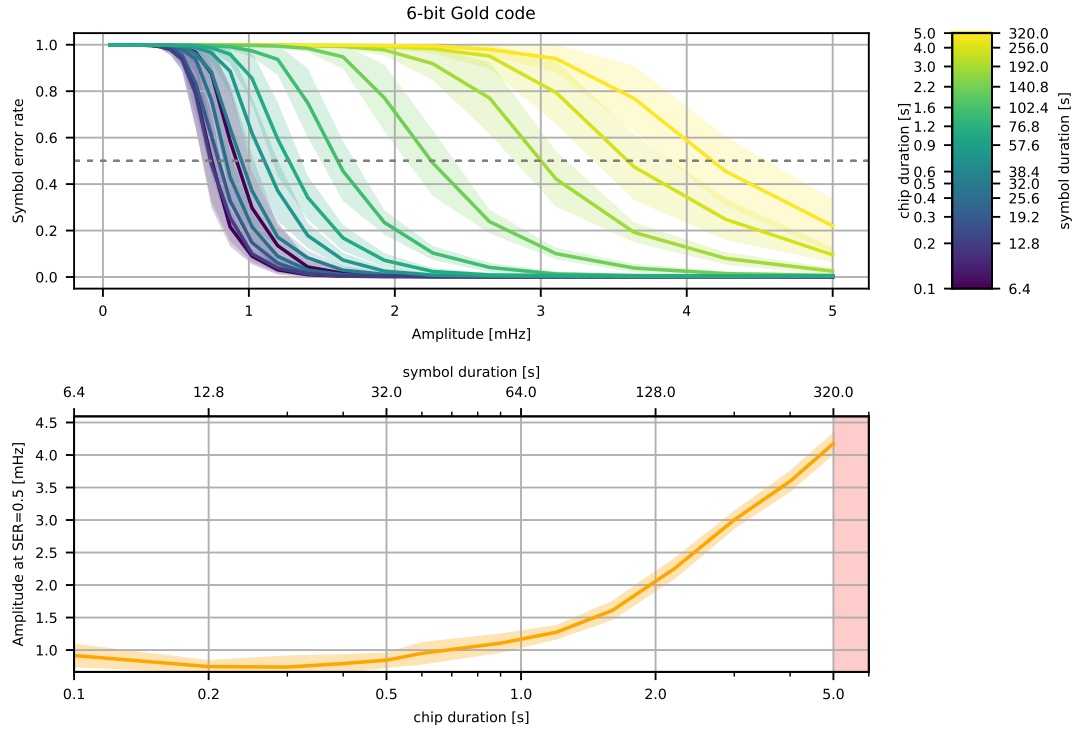

6-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]

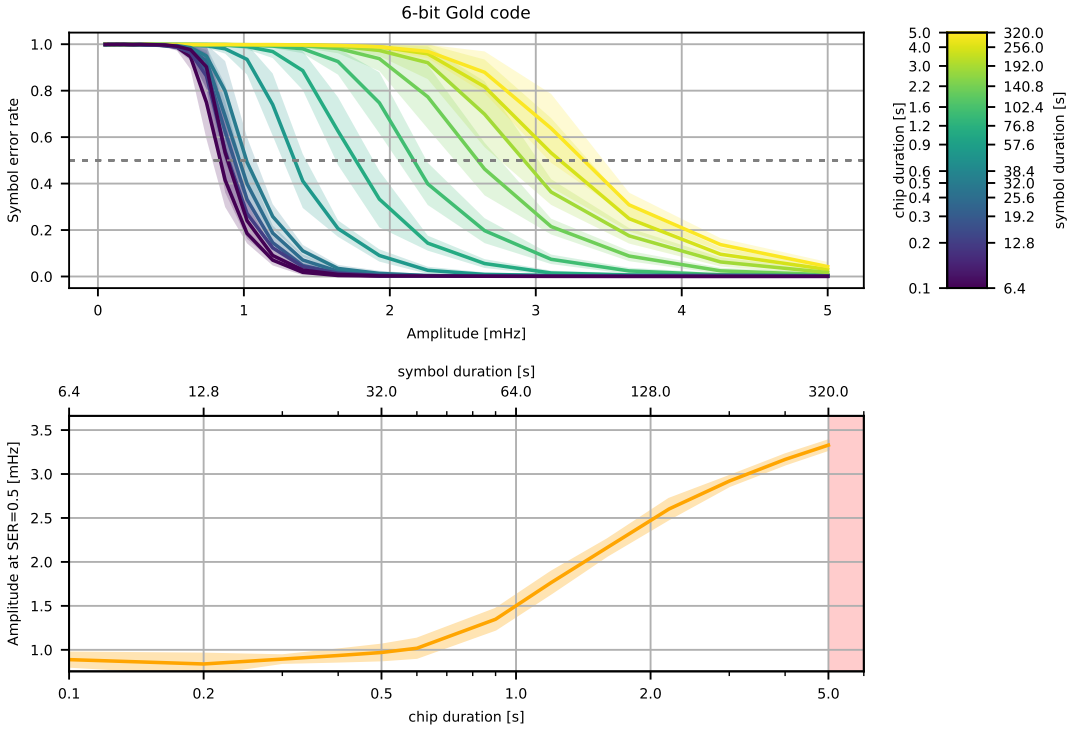


In [60]:
```
synth_files = [
        'data/dsss_experiments_res-par114-synth-run121-0-2020-04-11-17-50-31.json',
        'data/dsss_experiments_res-par114-synth-run121-1-2020-04-12-03-46-19.json',
        'data/dsss_experiments_res-par114-synth-run121-2-2020-04-11-18-44-15.json',
        'data/dsss_experiments_res-par114-synth-run121-3-2020-04-11-15-25-53.json',
               ]
plot_chip_duration_sensitivity(5, only_thf=4.0, files=new_files);
plot_chip_duration_sensitivity(5, only_thf=4.0, files=synth_files);
plot_chip_duration_sensitivity(6, only_thf=5.0, files=new_files, figsize=(7, 5))\
.savefig('fig_out/chip_duration_sensitivity_cmp_meas_6.pdf');
plot_chip_duration_sensitivity(6, only_thf=5.0, files=synth_files, figsize=(7, 5))\
.savefig('fig_out/chip_duration_sensitivity_cmp_synth_6.pdf');
```


5-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]


5-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]
symbol duration [s]


6-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]


6-bit Gold code
Symbol error rate
Amplitude [mHz]
chip duration [s]
symbol duration [s]
symbol duration [s]
Amplitude at SER=0.5 [mHz]
chip duration [s]

Appendix B

Demonstrator Resources

B.1 schematics and code

Appendix C

Demonstrator Firmware Symbol Sizes

libgcc.a(_udivmoddi4.o) (682B)
__udivmoddi4
reed_solomon / berlekamp.o (2250B)
Find_Roots (180B)
Modified_Berlekamp_Massey (532B)
compute_discrepancy (88B)
init_gamma (146B)
compute_modified_omega (106B)
scale_poly (72B)
add_polys (76B)
mul_z_poly (72B)
copy_poly (64B)
zero_poly (54B)
mult_polys (396B)
correct_errors_erasures (464B)
reed_solomon / galois.o (1136B)
gmult (238B)
ginv (28B)
gexp (116B)
galois_shift_inverse (294B)
galois_invert_binary_matrix (460B)
reed_solomon / ecc.o (100B)
rscode_init (38B)
rscode_decode (62B)
reed_solomon / rs.o (794B)
initialize_ecc (32B)
zero_fill_from (56B)
decode_data (116B)
check_syndrome (64B)
compute_genpoly (288B)
6387
primitive_polynomials (36B)
src / rslib.o (48B)
rslib_decode (48B)
driver
src / dsss_demod.o (3424B)
matcher_tick (684B)
group_received (960B)
cwt_convolve_step (128B)
gold_correlate_step (268B)
matcher_init (54B)
dsss_demod_step (804B)
dsss_demod_init (42B)
11161
decode_peak (58B)
11191
score_group (96B)
gaussian (104B)
musl / abs.o (28B)
abs (28B)
generated / dsss_cwt_wavelet.o (280B)
dsss_cwt_wavelet_table (4B)
cwt_ricker_69_window_7F3 (276B)
generated / gold_code_5.o (132B)
dsss_gold_code_table (132B)
musl / fabsf.o (42B)
fabsf (42B)
libgcc.a(_dvmd_tls.o) (4B)
__aeabi_idiv0
attributes
libgcc.a(_aeabi_uldivmod.o) (48B)
__aeabi_uldivmod
libgcc.a(_arm_addsubdf3.o) (888B)
__aeabi_drsub
__floatsidf
__adddf3
__aeabi_ul2d
__extendsfdf2
__aeabi_dsub
__aeabi_ul2d
__floatdidf
musl / __math_oflowf.o (36B)
__math_oflowf (36B)
musl / __math_uflowf.o (36B)
__math_uflowf (36B)
musl / __math_xflowf.o (142B)
__math_xflowf (74B)
fp_barrierf (34B)
eval_as_float (34B)
musl / expf.o (660B)
expf (564B)
top12 (26B)
eval_as_double (36B)
musl / exp2f_data.o (328B)
__exp2f_data (328B)
src / adc.o (754B)
adc_init (368B)
7256
adc_overruns
panic (16B)
adc_flt_buf
adc_flt_buf_ready_idx (4B)
DMA2_Stream0_IRQHandler (180B)
src / freq_meas.o (1274B)
adc_buf_measure_freq (944B)
func_gauss_grad (212B)
func_gauss (118B)
libgcc.a(_arm_muldf3.o) (596B)
__aeabi_dmul
libgcc.a(_arm_truncdfsf2.o) (160B)
__aeabi_d2f
cmsis / arm_rfft_fast_f32.o (938B)
arm_rfft_fast_f32 (78B)
merge_rfft_f32 (402B)
stage_rfft_f32 (458B)
cmsis / arm_bitreversal2.o (178B)
arm_bitreversal_32 (178B)
cmsis / arm_cfft_f32.o (4530B)
arm_cfft_f32 (372B)
arm_cfft_radix8by4_f32 (3158B)
arm_cfft_radix8by2_f32 (1000B)
cmsis / arm_cfft_radix8_f32.o (4108B)
arm_radix8_butterfly_f32 (4108B)
levmarq / levmarq.o (2732B)
levmarq_init (64B)
levmarq (1652B)
error_func (154B)
cholesky_decomp (488B)
solve_axb_cholesky (374B)
musl / sqrtf.o (408B)
sqrtf (408B)
generated / fmeas_fft_window.o (1028B)
fmeas_fft_window_table (4B)
fft_256_window_gaussian_16 (1024B)
cmsis / arm_rfft_fast_init_f32.o (76B)
arm_rfft_256_fast_init_f32 (76B)
cmsis / arm_cfft_init_f32.o (88B)
arm_cfft_init_f32 (88B)
cmsis / arm_const_structs.o (16B)
arm_cfft_sR_f32_len128 (16B)
cmsis / arm_common_tables.o (2464B)
twiddleCoef_128 (1024B)
armBitRevIndexTable128 (416B)
twiddleCoef_rfft_256 (1024B)
src / protocol.o (327B)
handle_dsss_received (292B)
src / system_stm32f4xx.o (56B)
SystemInit (56B)
src / startup_stm32f407xx.o (82B)
g_pfnVectors
Reset_Handler (80B)
Default_Handler (2B)
RTC_Alarm_IRQHandler
src / main.o (18761B)
assert_func (18B)
main (824B)
__libc_init_array (14B)
PendSV_Handler (16B)
SVC_Handler (16B)
BusFault_Handler (16B)
HardFault_Handler (16B)
SysTick_Handler (16B)
DebugMon_Handler (16B)
UsageFault_Handler (16B)
MemManage_Handler (16B)
NMI_Handler (16B)
oob_trigger_activated (156B)
iplt
plt
measurement_errors
spi_flash_setup (188B)
led_setup (72B)
clock_setup (344B)
debug_last_freq
freq_sample_ts
demod_state
update_image_flash_counter (312B)
jt_spi_flash_read_block (80B)
jtag_img (12B)
fw_dump (16384B)
8287
apb2_timer_speed
spi_flash_if_set_cs (48B)
spif
apb1_speed
sysclk_speed
apb1_timer_speed
auxclk_speed
apb2_speed
section .isr_vector
src / spi_flash.o (260B)
spif_init (128B)
spi_write (34B)
spi_read (30B)
spi_xfer (68B)
tinyprintf / tinyprintf.o (2430B)
ltp_format (1036B)
l2a (46B)
ll2a (46B)
lli2a (62B)
puitchw (404B)
a2u (102B)
ui2a (190B)
uli2a (190B)
ulli2a (274B)
a2d (80B)
src / serial.o (1194B)
NVIC_SetPriority (84B)
__NVIC_EnableIRQ (60B)
usart_dma_init (156B)
usart_printf (56B)
usart_dma_reset (92B)
usart_flush (108B)
usart_putc_nonblocking_tpf (30B)
usart_printf_blocking_va (52B)
usart_wait_chunk_free (42B)
usart_putc_blocking_tpf (30B)
usart_dma_stream_irq (94B)
usart_schedule_dma (228B)
usart_putc_nonblocking (90B)
usart_putc_blocking (72B)
src / dma_util.o (319B)
dma_get_isr_and_clear (300B)
6895
src / mspdebug_wrapper.o (1514B)
sr_jtdev_led_green (22B)
sr_jtdev_tclk_strobe (72B)
sr_jtdev_tst (26B)
sr_jtdev_tclk (26B)
sr_jtdev_tdi (26B)
sr_jtdev_tms (26B)
sr_jtdev_tck (26B)
sr_jtdev_connect (20B)
sr_jtdev_power_on (20B)
sr_jtdev_rst (26B)
sr_jtdev_led_red (22B)
sr_jtdev_vtable (68B)
mspd_jtag_init (104B)
sr_jtdev_tclk_get (52B)
sr_jtdev_tdo_get (52B)
sr_delay_inst (40B)
sr_gpio_write (124B)
gpios (72B)
printc_err (108B)
mspd_jtag_flash_and_reset (444B)
htole (24B)
sr_jtdev_default (16B)
sr_jtdev (16B)
src / gpio_helpers.o (288B)
gpio_pin_setup (288B)
mspdebug / jtaglib.o (3663B)
jtag_init (292B)
jtag_release_device (160B)
jtag_erase_flash (484B)
jtag_write_flash (490B)
jtag_write_mem (128B)
jtag_reset_tap (176B)
jtag_execute_puc (144B)
jtag_write_reg (218B)
jtag_halt_cpu (112B)
jtag_release_cpu (66B)
jtag_get_device (132B)
jtag_set_breakpoint (304B)
jtag_set_instruction_fetch (108B)
jtag_is_fuse_blown (68B)
jtag_ir_shift (150B)
jtag_dr_shift_16 (126B)
jtag_shift (190B)
jtag_tclk_prep (76B)
libsodium / libsodium.a(libsodium_la-auth_hmacsha512.o) (522B)
crypto_auth_hmacsha512_update (36B)
crypto_auth_hmacsha512_final (80B)
crypto_auth_hmacsha512_init (322B)
crypto_auth_hmacsha512 (84B)
generated / crypto_presig_data.o (449B)
presig_height (4B)
presig_domain_strings (20B)
presig_keys (75B)
musl / memcmp.o (92B)
memcmp (92B)
musl / strlen.o (120B)
strlen (120B)
src / con_usart.o (796B)
con_usart_init (132B)
DMA2_Stream7_IRQHandler (20B)
con_usart (644B)
libsodium / libsodium.a(libsodium_la-hash_sha512_cp.o) (672B)
crypto_hash_sha512_update (852B)
crypto_hash_sha512_final (80B)
crypto_hash_sha512_init (60B)
SHA512_Transform (1816B)
SHA512_Pad (228B)
6279
PAD (128B)
be64enc_vect (70B)
rotr64 (118B)
Krnd (640B)
be64dec_vect (74B)
store64_be (254B)
load64_be (352B)
musl / memcpy.o (1368B)
memcpy (1368B)
musl / memset.o (514B)
memset (514B)
src / crypto.o (478B)
oob_message_received (58B)
verify_trigger_dom (220B)
verify_trigger (116B)
debug_hexdump (72B)
libsodium / libsodium.a(libsodium_la-utils.o) (66B)
sodium_memzero (448B)
_sodium_dummy_symbol_to_prevent_memzero_lto (22B)

Appendix D

Economic viability of countermeasures

D.1 Attack cost

D.2 Countermeasure cost