Auditable Real Random Number Generator Hardware:

The `arrgh` Project

Benedikt Stockebrand

DRAFT (July 21, 2014)

Contents

Chapter 1

Introduction

1.1 The Security Impact of "Bad Random Numbers"

For decades the relevance of cryptography grade random numbers was largely underestimated by the cryptographic community. Popular textbooks either entirely skipped the topic or blissfully abstracted the inherent complexity of the topic away.

In 2013, when secret documents leaked by Edward Snowden claimed that the NSA had managed to publish a backdoored cryptographically secure[!] pseudo-random number generator (CPRNG or CSPRNG) through the NIST, this finally changed.

Why would the NSA go through such effort to make a backdoored CPRNG a widely implemented and used standard? The claims suggest that they considered it a rather effective way to gain access to people's encrypted communication.

The key reason is that cryptographically secure random numbers are used as key material (called "critical security parameters", or "CSPs" in FIPS140 lingo) for "proper" cryptographic algorithms, i.e. encryption and authentication. With the vast majority of cryptographers at least outside the "intelligence" and military communities focusing on the trustworthiness of the cryptographic algorithms, focusing on the random number sources they use is exceedingly promising for a whole range of reasons.

With the lack of attention by the cryptographic community, chances of discovery were tremendously lower than for a direct attack against the cryptographic algorithms proper. The expected lifetime of CPRNGs was also much longer than that of an encryption or secure hash algorithm.

Additionally, while most computers offer a range of different cryptographic algorithms to choose from, they usually only offer a single source for cryptographically sound random numbers.

These two aspects combine disastrously: Many Unixoid operating systems as of today (mid 2014) still use the outdated Yarrow algorithm despite the fact that the significantly improved successor algorithm called Fortuna has been published more than ten years ago.

Yet another problem is that with encryption and authentication algorithms, all parties involved know what algorithm is used and can refuse to do so. With random number sources, it is impossible for the recipient of some key material to determine if the sending side used a secure source.

Finally, computers and non-determinism don't mix too well. Software as such is inherently deterministic, and hardware must be made deterministic as well to be

of any use in computing. As a consequence however, "randomness" in computers is a scarce resource, leading to several more or less questionable approaches to compensate for this.

CPRNGs use whatever minimal amount of "randomness" they can get hold of as "seed" to their deterministic algorithms and then use some sort of cryptographic algorithm to generate vast amounts of "effectively random" output at high speed. Aside from hiding a possibly subverted source of bad "seed" itself, CPRNGs also introduce the additional risks of broken cryptographic algorithms and insecure implementations of otherwise sound cryptographic algorithms.

To get whatever minimal amount of "randomness" to "seed" their CPRNGs, real world systems frequently resort to even more questionable approaches. They poll "frequently changing" system attributes, measure timings on external events and similar. How reliable these approaches are however depends on the environment they are run in. There have been several reports about CPE (customer premises equipment) home routers in a "quiet" environment, for example with a single computer waiting for "the Internet" to become available, using their CPRNG during startup with such insufficient "seed" that their output is highly predictable.

In summary, dependable sources for random numbers are essential for any computing device used for security-critical purposes—no matter how long this aspect has been neglected by most of the security community.

1.2 Essential Terminology

To understand the peculiarities of HWRNGs and the context they are generally used in, a some fundamental terminology is necessary.

Entropy source An entropy source generates output that isn't entirely predictable from some non-deterministic phenomenon. The output may be analog or digital, and it may be biased and segments of its output may be statistically correlated to other segments of output.

Hardware random number generator (HWRNG) A HWRNG is somewhat similar to an entropy source in that it generates non-deterministic output. However, different than an entropy source its output is digital and the bits in its output are unbiased and not correlated to each other. HWRNGs are usually built by combining an entropy source and an extractor that "distills" the output from the entropy source by removing bias and correlation. As such, HWRNGs are a special subtype of entropy source.

Cryptographically secure pseudo random number generator (CPRNG) CPRNGs are somewhat similar to HWRNGs, but they work by taking some very limited amount of input from an entropy source and feed that as seed into a pseudo random number generator that is using cryptographic algorithms to produce vast amounts of output that appears to be random.

Hardware security module (HSM) To protect cryptographic systems from outside manipulation, HSMs are used. They are meant prevent physical manipulation, usually by some sort of self-destruct feature, as well as remote manipulation by restricting the access interface to critical components. HSMs usually contain implementations of various cryptographic algorithms, key stores for the various cryptographic algorithms, and CPRNGs using the cryptographic algorithms. They may also contain entropy sources.

This very last feature of HSMs, to use built-in entropy sources, will turn out to be the fundamental problem that this project tries to address.

1.3 "IT Security" and "Tamper Protection"

A huge problem that has a particular effect on HWRNGs is the industry wide confusion between "IT security" and "IT insecurities handling".

The general attitude that a device, component or other product is "secure" unless proven otherwise leads to HWRNGs being produced and sold which can't be audited. But an audit is the only way to ensure that a HWRNG doesn't have a backdoor: Testing its algorithmic function by feeding it some defined input and checking for the expected output result isn't possible, simply because there is no input and any output is, or should be, equally probable.

This immediately collides with the widely promoted, and in some cases even legally required, "tamper protection" of security related devices. The problem with "tamper evident" or "tamper resistant" (or "tamper proof", if you talk to the wrong marketing people) devices is that by their very design these devices can't be properly audited without breaking them.

At best it may in some cases be possible, using some industrial or scientific grade equipment, highly specialized knowledge and skills, and a tremendous amount of man power and funding, to investigate the correctness of a single device sample—and destroying it in the process. Auditing a device before actually putting it to use is generally impossible.

A hardware security module (HSM) used to store key information or holds whatever other kind of confidential state information has reason to be tamper protected. But this protection comes at a hefty price as it makes the legitimate user of such a device more vulnerable to any manipulation during the design, production and delivery of such a device.

HWRNGs however don't hold any state information, and as such tamper protection beyond the "tamper evident" level is nothing but counterproductive. Combined with the inapplicability of standard test procedures this makes the built-in HWRNG in an HSM a prime target for any product-level pre-deployment attack.

In summary, while HWRNGs should be used in connection with mostly any HSM, they need to be treated specially due to these unique—and essential—properties.

Chapter 2

Security Basics for Cryptographic Hardware

At this point it should already be evident that the security model underlying the design of many cryptographic hardware solutions is lopsided if not thoroughly flawed, especially when it comes to dealing with the HWRNG or entropy source they use. So this chapter starts with a classification scheme for attacks followed by some fundamental defense strategies.

2.1 Classification of Attacks

There is no simple, one-dimensional scale by which to characterize different kinds of attacks. Instead, several categories create a multi-dimensional space within which attacks can be classified. With a context-specific weighing of these categories can attacks then be rated on a one-dimensional scale again; however, there is no generic such weighing, and as such, there is no universally applicable way to assess the relevance of a given attack without the context of a specific attacker and target.

2.1.1 Rationality of Attack

What is the motivation for an attack? Or, as a former criminal investigator I've met once put it: "If I had to choose to deal with either a bank robber with an assault rifle, or a lunatic with a slingshot, I'd always pick the bank robber."

Defending against irrational attackers is notoriously difficult. Jealousy, religious or political zeal, the sheer desire to destroy something or some kinds of mental derangement aren't a rational reason to attack some computer system, but that doesn't mean these attacks don't occur. And it most definitely doesn't mean they are any less serious.

In the context of this study, irrationally motivated attacks are largely out of scope—not because they are considered irrelevant, but because they have to be dealt with at a completely different level.

2.1.2 Levels of Access and Goals of Attacks

Not a single category, but more a subspace of the entire space created by the categories listed here, are the various levels of access attackers may gain and, closely related, the goals they may want to achieve.

Comparing levels of access is frequently difficult and largely context-dependent. Gaining root access on a Unixish system is largely considered more severe than

gaining non-root access only; if the target however is a database server, then gaining "only" database administrator privilege is effectively all that an attacker may need.

Similarly, gaining passive or read-only access to communications or data may be considered a lesser level of access than gaining the capability to inject new or modify existing communication or data; depending on the ultimate goal of the attack these distinctions may make a huge difference or be effectively irrelevant.

As far as this study is concerned, it is generally best not to make any assumptions about these categories.

2.1.3 Personal Risk for the Attacker

An important category to take into consideration is the personal risk for the attacker. Put bluntly, breaking into a computer, or range of computers, in a country that doesn't have an extradition treaty with the country of residence of the attacker is significantly less risky than physically breaking into a local police station.

So international attacks, especially between countries that have less than a cordial relationship, bear significantly less risk on the attacker than attacks within the same jurisdiction. On the other hand, from the target's point of view, the risk of international attacks is significantly higher than of local attacks.

In a similar vein, the particular legislature has a huge influence on this category. If using somebody else's mail address as the alleged source of spam mail was treated as seriously as forging somebody else's signature, this would surely make a noticeable difference to the average spammer.

Reasonably rational attackers will generally try to minimize their personal risk. As a potential target this can be leveraged to some degree, by trying to increase the risk for an attacker: If suitable security measures imply that an attack requires physical access to the premises of the target, then this increases not only the risk of discovery and capture of the attacker but also adds burglary to the charges.

While these examples may appear somewhat far-fetched at this point, the point is vital to the security of cryptographic systems: If an attacker needs to gain physical access to an HSM or HWRNG or similar to subvert it, rather than doing so remotely, this makes a huge difference. A simple pushbutton that needs to be pressed to update the firmware of such a device is well worth the extra complexity.

2.1.4 Risk of Discovery

A similar but different category is the risk that a target may discover the attack. Many attack types depend on going unnoticed; once the target discovers or even only suspects an attack, the attack has implicitly failed.

When targets discover that their confidential communication is being tapped, they will resort to a different communications channel and the attacker will be unable to snoop on them any further.

On the other hand, if the goal of an attack is to disrupt the communication between targets, then the risk of discovery is effectively 1, or 100%, but that doesn't mean it'll stop an attacker with that goal.

For the purposes of this study, increasing the risk of discovery for the attacker is always desirable. At worst, it doesn't make any difference, but so far there is no scenario known where it actually is counter-productive.

2.1.5 Scale/Scope of Attack

Possibly the most important category of attack, at least if the claims of Edward Snowden are even vaguely correct, the scale or scope of an attack.

If a reasonably rational attacker has a choice between attacking a single computer or an entire product line while all the other categories are otherwise unaffected, then it is more than likely that such an attacker will opt for subverting the entire product line.

This fact is frequently glossed over by commercial security hardware vendors who put a huge emphasis on targeted attacks against individual or small groups of devices while neglecting the risk of all-comprehensive carpet-bombing-style attacks against their products.

2.2 Types of Attacks on Cryptographic Hardware

With regard to cryptographic hardware, a number of different attack types are considered potentially relevant. This list by its very nature not complete, nor do all of these types apply to all scenarios. In fact, when designing cryptographic hardware, a number of fundamental design decisions have to be made which have a tremendous effect on the applicability of the design in a given scenario.

2.2.1 Extraction of Confidential Data

There are basically two reasons why to use an HSM: One is to increase throughput beyond the point that can be done in software on general-purpose hardware, and the other is to provide additional protection to the key material and state information used—the confidential data, or "critical security parameters", or "CSPs" in FIPS140 terminology.

The underlying assumption with the latter is that attackers may try to gain access to the HSM in such a way that they can covertly extract the key material or state information without leaving any permanent traces, thus allowing them to pose as the owner of the HSM using the key material, or by decrypting communications using the acquired state information.

At worst, these attacks can be done remotely, using some kind of remote read access to the confidential data. Designs that force an attacker to access the device physically are considered significantly more secure because they increase the risk and cost for the attacker while reducing the possible scope of an attack. Devices considered most secure however don't allow any extraction of confidential data at all; they may allow the installation of data through a write-only interface and then use it for their cryptographic purposes, but they don't allow any extraction of that data at all.

As obivous as this kind of ranking is, it misses two essential aspects that by themselves lead to an opposite ranking.

The first one applies to HSMs as such: If the device itself is suspect, then the more restricted access is, the harder it is generally to detect any manipulation. While in the context of this particular attack type this limitation may be uncritical, it will recur in more critical ways through large parts of the rest of this section.

This threat is entirely irrelevant with HWRNGs, simply because they don't hold any confidential data. For that reason, protection through tamper resistance doesn't offer any protection to a HWRNG; it just makes it harder to inspect. On the other hand, since HWRNGs are inherently non-deterministic by their very nature, inspection of the device is the only way to ensure its integrity. Put bluntly, a HWRNG that is made impossible to inspect due to tamper protection can only be either blindly trusted or not at all.

2.2.2 Injection of Fake Confidential Data

Another threat closely related to the previous one is the possibility that an attacker injects manipulated confidential data into a device.

Again, there is a major difference between attacks that can be exploited remotely, and those that require physical access to the device. Again, this issue must be considered from an operational point of view: If devices can't be managed remotely, then the effort necessary to establish physical access for authorized operations may well negate the security benefits of a local-only access design.

The reasons *why* it should be possible at all to inject fake confidential data are twofold: For operational purposes it may be desirable to have a way to replace a broken device and installing the confidential data from (a backup of) the broken device. From a testing point of view, if fake confidential data can't be injected, then functional tests of the device are effectively impossible.

Again, this threat is effectively irrelevant to a HWRNG for the same reasons as in the previous section.

2.2.3 Installation of Backdoors in Operational Devices

All IT components should generally be considered buggy. If there is no way to install updates, the real-world consequence is that even devices known to be insecure are kept in use. However, if a device caters for the installation of updates, this feature can be abused by an attacker to install a backdoor.

With regard to HSMs one might reason that this scenario is rather impractical, simply because an attack towards the computer that is using the HSM is far simpler. However, once such an attack has been successful, it is particularly unlikely to get noticed even if the computer installations used are subject to periodic audits.

Implementing features in hardware rather than software may make it more difficult to execute the actual replacement, but similarly, once the modified hardware is in place it is exceedingly difficult to discover

2.2.4 Replacement of Operational Devices

If installing a backdoor in an already operational device is time consuming, difficult, and generally involves a high risk for the attacker, an obvious alternative is to replace the device with a manipulated one instead.

When it comes to HSMs, this is somewhat difficult due to the confidential data stored in the device. An attacker has effectively two options: Try to transfer the data somehow—which can be made difficult by design—or rely on the owners/users of the device that they assume some sort of glitch, re-install the confidential data themselves, and resume operation with the replaced device.

With HWRNGs however, even this is not a problem for the attacker: Since the device doesn't generally hold any confidential data, all that may be noticed by the owners/users is a very short loss of service.

In other words, if an attacker intends to replace a crypto hardware device, a HWRNG-only device is a particularly promising target.

2.2.5 Targeted Distribution of Corrupted Devices

All attack models so far assume that a device is attacked after its deployment. However, replacing or modifying devices before they are actually delivered to the target opens a whole new range of possibilities—to the attacker.

The advantages for an attacker are plentiful: These attacks are inherently remote while still giving them full physical access to the device, minimizing effort and personal risk while at the same time providing them with a longer timeframe for their operation.

The major downside for an attacker is that a newly delivered device may be subject to some additional scrutiny by the receiver, but the way many commercial HSMs these days focus on the previous types of attacks it is usually feasible for an attacker to pass this additional hurdle.

2.2.6 Corruption of Entire Products or Product Families

All attack models so far assume a targeted attack at individual devices. However, attackers who have successfully managed to subvert a device design to a point that their devices pass the initial scrutiny on delivery, might as well see that they replace all devices of that type, rather than individual ones.

The key risk with this attack is that the risk of discovery increases linearly with the number of devices subverted. However, since the attack usually happens during manufacture, rather than during shipping as might when attacking individual devices, this risk is frequently so small that it may be acceptable to an attacker. Furthermore, if there is no "clean" reference device available to a user/owner to compare a compromised device with, the risk in a way decreases again.

This level of attack can be pushed even further if a manufacturer uses some underlying designs for entire product families. If all the crypto products of a vendor use the same chip core for their key algorithms, introducing a vulnerability into that core can make the entire product family vulnerable; if multiple vendors use the same core design from an external source, even multiple manufacturers can be affected.

2.2.7 Slipstreaming Corrupted Devices

A somewhat hybrid attack that combines aspects of the previous two attack types is to slipstream an attack into an already existing product line.

From an attacker's point of view this has the advantage that the first, uncompromised specimen will receive the most intense scrutiny, while the devices sold later on will be assumed trustworthy with regard to untargeted attacks.

This is important with regard to public scrutiny, where new device types may receive significant extra attention; it is however even more important in contexts which are security sensitive/aware enough to do in-depth certification based on individual sample devices.

2.2.8 Standardization Level Attacks

The recent allegations that the NSA have actively manipulated the NIST to standardize a CPRNG algorithm demonstrates a yet broader level of untargeted attack.

The primary difference between corrupting a product line or product family and this attack is its even broader scope. With the algorithms being publicly known however, there is an increased risk of additional attackers identifying the weakness by cryptographic analysis of the published design: While cryptographers have publicly voiced concerns about the security of the design, the standard was still established and put into implementation.

The implications of the attack are even more troubling: It shows that standardization doesn't necessarily ensure the level of security claimed by the standardizing body. It also shows that peer review can be circumvented at this level of attack by

leaving the burden of proof that a design is insecure to the reviewers rather than the developers of the design.

2.2.9 Political Level Attacks

Finally, history has shown that attacks have been repeatedly at least attempted at the political level.

Probably best known with regard to this was the attempt of the NSA to legally restrict the use of publicly available cryptography to the Clipper chip design, which contained a backdoor for the NSA. While this attempt eventually failed, strong cryptograpy is still subject to legal restrictions in many countries, frequently still rating it as military-grade arms and ammunitions.

These, and possibly the existence of "secret laws" and "secret courts" to enforce them, may lead to situations where hardware vendors are covertly forced to install hidden backdoors in their products. The end effect of such attacks is similar to the standardization level attacks except that discovery of such backdoors, unless done in an exceedingly simplistic way, is next to impossible unless the details of the product are disclosed.

At a more mundane level, legal requirements enforcing some sort of "protection level" which may improve protection against targeted attacks to some degree but which make broad scale attacks almost impossible to discover, are used in many legislations—not necessarily by intention, but by too narrow a focus on targeted attacks.

2.3 Defense Strategies

As diverse as the attack types are the possible defense strategies. Commercial products as of today strongly focus on targeted attacks at the expense of addressing broader scope untargeted attacks. However, there are defenses against these attacks as well.

2.3.1 Technical and Economic Defenses

While commercial products tend to focus solely on technical defenses with the intention to make it practically impossible for attackers to break into their products, broad scope attacks may be more easily countered by employing economic defenses.

The rationale here is that untargeted attacks are currently most convenient for the majority of attackers—with regard to crypto hardware and also well beyond. The risk of discovery is manageable, especially so if the backdoor can be disguised as an accidential product fault rather than an intentional attack, the possibility of "preemtive strikes" is enticing, and most importantly, large scale attacks are economically much more interesting than targeted attacks.

As a consequence, untargeted attacks must be considered a major concern with any security architecture. Trying to respond at the technical level only is likely to be futile; combining those with leveraging the economic aspects of untargeted attacks does however promise a significant increase in resilience against untargeted attacks.

2.3.2 Burden of Proof and Peer Review

One of the underlying reasons for the sorry state of what is generally touted as "IT security" is the fact that, despite its name, most of the time it isn't. Large parts of

the “IT security” industry should really be called “blatant IT insecurities coping” industry instead.

In most of IT the general assumption is that “unless it’s proven beyond doubt (preferably by being actively exploited) that there is an insecurity, any vendor’s claim about the alleged security capabilities of their product is to be taken at face value without further doubt”.

From personal experience, prospective customers looking crypto hardware who ask for detailed technical information are generally fed glossy brochures which make all sorts of claims but show little technical detail, let alone any option to validate the claims made, let alone to audit the devices for conformance with whatever specification. More insistent requests are normally answered with more glossy brochures claiming various certifications and still no more details. If neither the customer nor the sales representative give up, the next step usually involves statements concerning trade secrets and the fear of falling victim to patent lawsuits, and possibly an offer to some alleged detail information provided that the customer first signs a non-disclosure agreement which denies the right to make the discovery of any insecurity publicly known.

What is really needed is a generally adopted attitude with regard to “IT security” that any product has to be assumed seriously insecure unless the developer or vendor can provide some sort of evidence that the product is sound; the related legal term is that the vendor has to bear the “burden of proof”.

This also means that the industry needs a significant number of qualified people to spend significant time verifying the claims of the vendors; the only good news is that the few people these days spend time on painstakingly reverse engineering product samples without access to the background documentation might become more efficient once they got access to this kind of information.

2.3.3 Diversity

Possibly the most effective, but economically also the most difficult, line of defense is to increase the number of products and product variants an attacker has to subvert.

However, IT is one of the industries most susceptible to the economics of scale; with the tremendously large ratio of initial development cost to reproduction cost, IT is particularly prone to few large vendors dominating the market.

The open hardware/open source/open design communities may be an alternative to the established vendors, although anything short of a ready-to-use product from a “reputable” source will only read a niche market in the short run.

Outside a context driven by short-term economic goals, designs that are intentionally made simple to modify are considered the most effective to defend against untargeted attacks.

2.3.4 Auditability

A second countermeasure against untargeted, large scale attacks is a design that is intended to be auditable by third parties and reasonably equipped and qualified end user.

This approach implies that a design is done in such a way that individual devices can be tested non-destructively, repeatedly, and while already deployed. The nature of most tamper-resistant or even just tamper-evident designs implicitly makes a proper audit infeasible; it remains to be seen how these two defense mechanisms can be combined.

Another problem with auditable designs is the fact that they are vulnerable to unwarranted patent claims, especially in jurisdictions where the victim of such a claim will take significantly higher economic losses by fighting the claim in court than by agreeing to an unwarranted out-of-court settlement. As experience with open source software has shown, open solutions are by their nature somewhat protected against this.

If auditability is used as a defense against large scale attacks, this implies that a design is the better the quicker it can be audited with the least necessary tools and skills. This however contradicts the diversity approach, since only designs with a non-negligible number of devices deployed can possibly be subject to a sufficient level of scrutiny.

2.3.5 The KISS Principle

To make a design easy and quick to audit, it has to be designed to be simple, as required by the more general KISS ("keep it simple, stupid") principle known throughout IT in general.

The first implication here is that a design that just does what it is required to do the job is best. For example, while the Cryptech project currently tries to build a full-blown HSM, the `arrgh` project only focuses on a HWRNG. If only a HWRNG is needed, then the `arrgh` design is superior because of its simplicity: No hard-to-audit FPGA, significantly reduced reliance on—possibly subverted—development tools, and the option to assemble an individual device from scratch. On the other hand, if the other features of an HSM are actually needed, then the Cryptech design is obviously not just the better, but effectively the only choice between the two.

From this it already follows that diversity, not by design but by feature set, is another valuable defense against any kind of attack.

It also follows that modular designs, where only the features needed are actually installed, provide more resilience than an all-features-in-a-single-box designs.

2.3.6 Modularity

Aside from the feature-oriented modularity explained in the previous section, modular hardware designs are provide significantly improved protection against untargeted attacks.

With a modular hardware design, auditing is tremendously simplified: Individual components can be tested separately, suspect components can be connected to a testbed that makes any possible manipulation obvious and quite generally, modularity makes an auditor's work significantly easier.

Another advantage of modular designs is that it allows for the modification of individual modules or subsets of modules from a design, thereby improving diversity.

2.3.7 Choice of Components

A surprisingly complex topic is the choice of components used in an auditable design. Not all commonly used components are suitable for an auditable and diverse design, and sometimes it is necessary to balance functionality and security at the component level.

Component selection criteria are considered universally helpful to increase the difficulty of manipulations at the component level:

Cheap, general purpose, high volume components Designs that use only or at least mostly cheap, general purpose, high volume components are economically more difficult to subvert than designs that heavily rely on highly specific components.

Attackers who try to manipulate such components have several problems: Untargeted attacks are expensive due to the high volume and carry an increased risk of discovery due to breaking unrelated products using the same component for a different purpose. Targeted attacks may still be viable, but since general purpose components are more likely to be modified than highly specialized ones, diversity will make such attacks at least more expensive.

Multi-vendor components If equivalent components are available from multiple vendors, again this makes them harder to attack: This makes for trivial diversity, simply by choosing components from whatever vendor.

If nothing else, attackers might try to promote their compromised components by selling them well below price; however, this again makes such an attack economically difficult.

Simple components Simply put, it is much more difficult to subvert a 1kΩ, 5%, 250mW carbon film resistor than an FPGA chip, not only for the reasons stated above, but because such a resistor is limited by its two terminals in the possible attacks it may execute and because these components make for much easier auditing.

Generally available components If components are easily obtained through a range of distributors, or even better found in every enthusiast user's junk parts bin, then more targeted attacks during the delivery of these components to the assembler become more difficult. Additionally, if components obtained elsewhere can be used as reference for an audit, this increases the chance to discover any manipulated ones.

Components with many alternatives Components that can be easily replaced by similar but different components—small signal transistors immediately come to mind—again increase diversity and generally amplify the effects of the previous criteria again.

While all these criteria are sometimes difficult to apply, they universally improve the security of a product.

Another question however is not as simple: The decision between through-hole devices and surface mount devices, plus the choice of package type for ICs.

Through-hole (THT) components These components are popular with entry-level home electronics amateurs and allow for reasonably straightforward home etched PCBs and even perfboard implementations. However, being fairly large they lead to large PCBs, which in turn may be operationally impractical. In some cases the size may be sufficiently large for an attacker to hide components underneath, especially bare die chips.

Unfortunately, with regard to ICs, some ICs are simply not available in THT packages, either due to their pin count or due to product portfolio decisions of the manufacturers.

Socketed vs. soldered-in THT ICs Especially with regard to ICs, THT components have another peculiar advantage: They can be socketed rather than soldered in. This makes auditing significantly simpler: Components can be removed from their sockets for easier testing of both the component and the remaining board and it is possible to inspect the PCB underneath the IC.

Socketed ICs have disadvantages, though: Targeted attacks aiming at replacing such an IC are simplified, the sockets add to bulk and weight of the entire board, the contacts between IC and socket are known to sometimes fail and the IC may be shaken out of the socket due to heavy vibration.

Surface mounted (SMD) components Throughout the electronics industry, surface mounted components are much more popular than THT components due to the simplified manufacturing processes, smaller board sizes and a much wider range of components especially with regard to high pin count components.

However, many home builders are uncomfortable soldering SMD, there is no reasonable way to build perfboard designs and, most troublesome with regard to auditing, SMD ICs generally can't be socketed.

SMD component sizes SMD component come in a wide range of sizes; the larger ones are easily soldered by home amateurs—once they've overcome their fear of them—and reasonably easy to audit while still on board. Sizes of less than 0603 two-terminal packages however should only be considered if board size is at a premium and auditing is less of an issue as it should be in the context of this paper.

Large SMD components like ICs, possibly large electrolytic capacitors and similar devices should be avoided if it is deemed possible to hide a bare die or similar underneath it.

Leaded vs. unleaded SMD ICs (as in "Leeds United", not "Led Zeppelin") Generally speaking, only leaded SMD ICs, preferably SOIC (50 mil pin pitch), possibly SSOP (usually 25 mil pin pitch) and if otherwise unavailable QFP (down to 16 mil pitch) should be used. Unleaded packages, packages with a large heat transfer pad underneath and BGA (ball grid array) packages should be avoided if at all possible since they can't reasonably be soldered by a home builder and are mostly impossible to audit.

If for some reason, BGA or similar are unavoidable, then particular care must be taken to modularize the design in such a way that these components can be tested in situ.

Bare dies These are even worse than BGA devices, require special machinery to install and as such have all the disadvantages of BGA plus some more. There should be no need to use these at all.

The problem of hidden bare dies or similar underneath ICs and other large components may possibly be mitigated by two alternative approaches.

Milled cut-outs underneath components Given conventional (usually FR4 or possibly FR2) PCB material it may be possible to ensure that nothing can be hidden underneath large components by cutting/milling holes underneath them.

At this point it is yet unknown if this affects industrial production. What is known is that the space underneath ICs is often densely populated with traces which will have to be routed elsewhere in this case.

Translucent flexible PCBs A possible alternative may be the use of translucent, flexible PCB carrier material. These should be too thin to hide a bare die underneath and allow visual inspection through the carrier material anyway.

At this point again it is yet unknown if this is practical, both for industrial manufacture and home building; there is however evidence that this may be feasible.

Both of these approaches however do share a common disadvantage: They are mechanically weaker than a solid FR4 board without cutouts.

2.3.8 Do It Yourself/Modify It Yourself Designs

A rather unusual way to maximize the effects of auditing and diversity is to involve the end user/owner in the design and production process.

If devices are provided as kits for users to assemble them themselves, this ensures that each devices receive significant auditing after shipping to the user. Manipulations like hiding a bare die beneath a soldered-in chip become effectively impractical.

Providing only the bare PCBs and expecting the users to shop for the remaining components themselves may additionally thwart more targeted attacks within the supply chain.

Providing only PCB blueprints and expecting users to manufacture even the PCBs themselves may mitigate even more attacks, however at significant economic costs (if users individually order the boards to be custom manufactured) or less of a reach of the project because home users won't etch PCBs themselves.

The biggest advantage of products which require end users to finish them however is that such products may encourage end users to modify the given design, again significantly increasing diversity.

2.3.9 Development Toolchain

Besides the consideration so far, which were either rather fundamental or hardware centric, software is rather important. Generally speaking, the less different kinds of tools are needed, and the more alternatives are available, the more resilient will the design be from attacks directed towards the development toolchain.

There are at least these tools needed to design and build a cryptographic hardware module:

Hardware design tools While circuit boards may in theory still be designed using paper for the schematics and transparent film for the board layout instead of a computer, doing so is tedious and doesn't produce the data needed for industrial manufacture.

The key requirement to the tools used here is that they are available to as many end users as possible. In-house only and prohibitively expensive tools are not an option, open source—which can be audited—is preferred.

Diversity is also desirable, but since there are generally no interchangeable formats in use, this is largely limited to the tools to verify the Gerber files output at the end of the toolchain. With multiple Gerber viewing tools, diversified audition of the board design is tedious but possible.

Firmware development and installation tools Components that need to run some embedded firmware require a toolchain to develop and handle that firmware. Since binary files are significantly more to audit than a set of Gerber files for a board design the requirements in this case are significantly more demanding than with the hardware design tools.

Development tools usually include compilers and possibly linkers depending on the kind of hardware component to be programmed. For installation it is usually necessary to use some sort of programming tool.

For auditing an extraction tool to retrieve the binary running on the component is needed, plus multiple disassembly tools to ensure that the binary correctly implements the original source code.

Host side driver development and installation tools While the host that a cryptographic device connects to is generally out of scope of this project, the software interface between the two cannot be ignored at this point.

Generally the same considerations apply to driver development as they apply to firmware; however, deployment is usually done through some operating system specific means rather than some sort of physical installation interface.

Finally, all these tools should be considered in the operating environment they are used on—from computer hardware to operating system to networking between multiple computers possibly used.

2.3.10 Physical/Tamper Protection

Finally, what about the physical protection, or tamper resistance, or even "tamper proofness" of commercial products?

First off, some level of tamper protection may be required for whatever legal reasons as mentioned before. However, since these legal requirements can't really be changed with this paper, we mostly ignore these.

As the works of people like Ross Anderson have shown, there is little to expect from physical protection measures against a serious attacker. The no-read flags with some microcontroller types (called "lock bits" with Atmel for example) are known to provide little defense against a reasonably professional product pirate, either. In summary, any sort of tamper protection is about as good as a cheap lock on a good bicycle: It tells the world that you want them to leave their hands off, but it provides protection only against an opportunistic attacker, not a really determined one.

What's worse, and what has been explained before, physical protection puts way too much emphasis on targeted attacks; with untargeted, large scale attacks in mind they tend to be counterproductive because they make several other defenses against such attacks more difficult if not impossible.

2.4 Interface Design

2.4.1 Restricted Remote Interface

2.4.2 Update Security

2.4.3 Cross Checking and Minimized Mutual Trust

Chapter 3

Theoretical Background

3.1 Noise, Entropy, and Randomness

3.2 The Relationship between HWRNGs and CPRNGs

3.3 Testing Methodologies

Chapter 4

The arrgh Design

4.1 Fundamental Design Decisions

4.1.1 Project Scope

4.1.2 Threat Model

4.1.3 Software vs. Microcontroller vs. FPGA vs. ASIC

4.1.4 Entropy Sources

4.1.5 Output Interfaces

4.2 Hardware

4.2.1 Overall Design

Module Outline

Noise Interface

4.2.2 Modules

Printed Circuit Board (PCB) Considerations

Power Backplane

12V Step-up Converter

Noise Generator/Entropy Source

Microcontroller and Outbound Interface

4.3 Microcontroller Firmware

4.4 Computer Side Driver

Chapter 5

The `arrgh` Implementation

Chapter 6

Auditing an `arrgh` Board

Chapter 7

Conclusions and Future Works