COINS summer school 2017 - reflection report
Secure cryptography on constrained devices

Andreas Lindner
KTH Royal Institute of Technology
andili@kth.se

September 2017

1 Introduction

Currently, we are in an era of increasingly connected and pervasively deployed and miniature electronic devices. Exactly this increased connectivity directly translates to increased attack options for hostile entities. Therefore, implementers face severe security challenges when working with the little resources available in sensor or actuator nodes. While typical application processors have gigabytes of RAM, caches, pipeline architectures and multiple cores, many architectures for embedded devices are very specialized and sometimes even lack a real stack for variables.

The following report is a summary and reflection of all the sessions of the COINS summer school 2017, relating topics of secure cryptography to my PhD project. Since the sessions covered different topics, I will address them in separate sections. The topics covered span from hardware trojans over side-channel cryptoanalysis to crypto software implementation, secure software updates, an application to car module updates, execution protection and an application thereof, and finally reliable side-channel security using three different approaches.

2 Detection of hardware trojans

On the first day, Paris Kitsos introduced hardware trojans and presented an approach to detect them followed by a practical experiment using an FPGA board. As part of the introduction, Kitsos provided two examples of hardware trojans, namely a combinatorial and a sequential hardware trojan.

While the combinatorial trojan consists of an AND-gate-tree to detect a pattern and change operation, the sequential trojan somehow detects a pattern and then enables a separate circuit with no outputs. This separate circuit is a preloaded shift register and its malicious effect after activation is simply unwanted power consumption. As a detection mechanism Kitos introduces Ring

Oscillators to sense the presence of additional or changed circuitry. For this purpose, strings of them are placed over the target circuit and then their frequency of operation is the quantity we take to characterize inherent circuit properties.

In the practical experiment, we were provided with an FPGA development board. On our computers, we compiled the project with the features mentioned above. Additionally, the project enabled us to control which frequency indication is sent to the display and which ring oscillators are enabled. A serial port over USB allowed us to send input to the target circuit, which was plaintext for AES encryption in this case.

Since I want my research to go down to the hardware level as practically possible in my time, I found this session inspiring. Realistically and most likely, I will not produce results with gate-level trojans or similar but I think I should be aware of some work in this area. In this way, I am able to understand which assumptions of a system built on top of such hardware are sound or in general how realistic they are. Furthermore, it eases reading related work and literature studies related to this.

Question Why do we need to enable the ring oscillators in the experiment individually and why are they not just running all the time?
Answer The running ones would interfere with the ones actually used for measurement and thereby disturb the measurement itself.

Question Who is the attacker, i.e., where is the attacker in the "chip production chain"?
Answer For ASICs, they are in the mask production; for FPGAs, everybody capable of changing the bitstream while it is loaded on the FPGA (this is possible and articles describing it exist) or even before.

Question Why and how does the additional hardware affect the ring oscillator?
Answer The factors affecting the frequency are current flows, operating voltage changes, additional heat, electro magnetics, and more. Note that many and shorter ring oscillators are better for detection, and small trojans are very hard to detect.

3 Side-channel cryptoanalysis

On the second day, Lejla Batina started with examples of general side-channel analyses to show the application areas and the basic principles and goals of attack. Then we had to implement such an attack on provided EM traces, sufficiently many to recover a small key.

Among the everyday areas of attack, medical or patient RFID, electronic passports, and set top media streaming (DRM) are the examples that everyone has heard of at some point. These either gain publicity because their users are victim or because piracy enables users to save money. What they have in common is that in all cases the corresponding devices try to protect data in some way

But because of different kinds of interests, they get under attack and because their implementations are not sufficiently secure the attackers succeeded.

The truth is that perfectly secure implementations are infeasible. Furthermore, developing such devices has a budget associated of course. In the sphere of resource-constrained devices the resources available for development are typically constrained as well. This is at least true for consumer products like mass-products of some sort. To make matters worse, these devices are typically accessible and not completely locked up. In the example of the set top box, an attacker can have access to such a device like any honest user.

Further examples of attacks involve backdoors in chips, insecure FPGA bitstream loading, and poorly implemented crypto protocols. In may instances of the latter, an insufficient treatment of randomness nor insecure key operation lead to the leak of sensitive information. This enabled scientists to break the keyloq technology for keyless car entry for arbitrary cars using this technology. After they successfully recovered the master key, it is enough to eavesdrop in 100m distance and collect 2 RF traces. Another example is Sony's PS3, where a supposedly random number is implemented as a constant.

Concrete side channel attacks span from measuring attacks like cache attacks, which normally are implemented as a timing attack, acoustic attacks and even data injection. Examples of cache attacks are FLUSH+RELOAD and cachebleed. Acoustic cryptoanalysis is presented in the proceedings of GST13 by Shamir et al. for example. On the other hand, the rowhammer attack enables the injection of data in adjacent memory rows because of a design flaw or bad tradeoff decision in DRAM implementations.

The basic idea of cryptoanalysis is to extract "one sub-key at a time". This enables a feasible key search in an attack exercise. In the case of power consumption as subject, static CMOS is interesting for two reasons. First, it is the prevalent chip technology nowadays. Second, its instantaneous power consumption depends on "processing", i.e., bit changes. This enables the analysis with two simple power models: the hamming distance model and the hamming weight model.

A successful analysis relies also on a good choice of input plaintext. It has to be well distributed to avoid the early introduction of unwanted correlations, which are just bias from the sample selection. It can be useful to start with additional, possibly simpler, investigations to learn more about an implementation, e.g., by spectrum analysis. This can be key lengths, the crypto algorithm used, and synchronization for correlating traces.

In the practical experiment, we were asked to implement a small cryptoanalysis based on EM traces of a running PRESENT cipher encryption. We had to first implement the first application of XOR and a 4-bit S-box. Then, we could use this to compute 16 hypotheses for the resulting value. After implementing and applying a hamming weight function to the resulting values, we could obtain the hamming weights that we expect to impose a proportional instantaneous power consumption at the same time somewhere in all the power traces. This was calculated by translating the traces together with the 16 hypothetical hamming weights into 16 correlation index traces. The highest value among all

the traces indicated the time when the computation is dependent on the result and the trace where it occurred revealed the key.

In my group, other colleagues were studying cache side-channels in the lines of cachebleed. Proving side-channel freedom for a piece of software using processor models, or extending a processor model for a certain side-channel are both very much related to my research and also possible subtopics.

Question Does the "nice shuffling" of the S-box help to sort out wrong key assumptions in the simplified example presented? Example: If the LSB of the S-box output was dependent on, say, the MSB of the plaintext, would then the wrong power traces get selected?
Answer Yes! According to Shannon, a good secure design is based on confusion and diffusion. But there is a certain irony in it, since this helps side channel analysis with sorting out wrong hypotheses. Sometimes it makes sense to weaken Shannon's confusion and diffusion to strengthen an artifact against side channel analysis.

Question Why are the measurements for the lab sheet taken with an antenna on top of the chip, which causes increased noise?
Answer They are actually not power measurements but EM measurements. This simply causes more noise.

Question Who are the customers of the dutch security company?
Answer They consult manufacturers of car keys, smartcards, broadcasting set top boxes for example. This consulting includes helping their customers, which could also be subcontractors for developing the aforementioned, for getting certifications and related specialized work they require for their products.

4 Optimizing crypto software

On the third day, Peter Schwabe introduced crypto software and its assembly-level optimization simultaneously considering side-channels by examples. Later, we had to optimize a microcontroller implementation of ChaCha20 ourselves. Crypto can be used for encryption and authentication of data. The idea of encryption is that no attacker can read while authentication should prevent attackers to change a message. With encryption and authentication, no attacker should be able to read or write.

Crypto optimization is different from general optimizations, since want to optimize under the constraint of security, e.g., to prevent timing leakage. One approach for this is "constant-time" code, which means that the execution time of the code has to be independent of secret data. For this purpose, a branch on secret data can be eliminated and assignments implemented as arithmetic operation. The simple coding rule about this says "no **if** based on secret data". In the presence of caches in the processor or memory system, secret dependent memory accesses can be used by an attacker like the most basic AES cache

timing attacks show. The simple coding rule about this says "do not access memory at secret-data-dependent addresses".

For my PhD project, the applied prevention of side-channel attacks in the development are interesting. This session helps me to identify side-channels or implementation flaws, which can be exploited, and at the same time I am prepared for developing and proposing countermeasures. These would then be implemented on the system level and possibly proven to be correct.

Question Why is ChaCha20 used by Google?
Answer AES-GCM (counter mode) is essentially also a stream cypher, as well as RC4 is a stream cipher. Both have been the only stream ciphers included in TLS before ChaCha20 was included. ChaCha20 is very simple to implement securely – more or less any straight forward implementation is also secure. Google's Adam Langley was pushing for it since AES is hard to implement securely on all platforms. It was even completely lacking on some platform's implementation of TLS. Then ChaCha20 finally got included in TLS and used by Google.

Question What is the difference between ChaCha20 and Salsa20?
Answer Salsa20 was the winner of the original crypto competition. ChaCha20 is an improved version for performance and it is probably slightly more secure, i.e., comprised of less attackable rounds. On the other hand, Salsa20 has undergone more checking in terms of cryptoanalysis during the competition.

Question Do we "optimize securely" in the lab exercise?
Answer Yes, everything is just arithmetic and we do not deal with lookup tables. ChaCha20 is straightforward to implement mostly side-channel free.

Question Why do we have stream ciphers?
Answer They are cheaper than block ciphers. AES in counter mode has a decrypt function while a stream cipher only requires a key string using encrypt. This already demonstrates that block ciphers are not designed for this. A pure stream cipher designed for just this purpose is better and faster. Furthermore, encrypting a message as a block has not the "flipping a bit in the ciphertext, flips also the corresponding bit in the plaintext" behavior.

5 Secure software updates

On the fourth day, Justin Cappos gave two sessions. The first one was on software update security starting with a motivation for software updaters as good target. Step by step, he introduced new details about the inner workings of the server and client side of update operations. At the same time, we discussed more and more subtle attacks.

Software updaters are good targets, because they:

- run as root,

- update / change software (which is mostly the overall goal),
- make traditional defenses (firewall/IDS) ineffective,
- make attacks ubiquitous (since applicable to all kinds of devices), and
- make attacks appears benign (updater misconfiguration, etc.).

In general hacker organizations use the same tools/burner servers/libraries/attack models, again and again, which makes these assets candidates for fingerprinting for attribution. A good thing on the defenders' side i that it is not clear to attackers which burner servers (or other assets) have been compromised; the attackers do mistakes there or reuse their assets instead of changing everything for each attack in order to erase all traces. An example for this are Ukraine attacks and the US election.

When we think about the software repository model, where several clients download files from a central server; what can an attacker possibly do here?

- man in the middle (during transmission, DNS, HTTP/FTP trafic),
- server (file storage) compromise: replace data desired by client,
- denial of service (deny a file),
- slowly send data,
- endless data (which works if nobody knows size of the desired data),
- ZIP bomb (or also JSON or other formats' bombs).

The effects for the endless data attack involve for apt that the system runs out of memory, freezes up and crashes. Yum fills the disk until there is no space left. For the ZIP bomb, 42kb of transferred data become petabytes on the client instead. Alternatively, processing takes huge amounts of time.

The way how software updaters split metadata and software packages enable attacks like "duplicate files in ZIP archive". In this case, a file can be listed multiple times, where the software update client will check only one of them, which is maybe not the one that is applied later. Furthermore, manipulating version numbers can cause:

- freezing by providing the same version over and over ("there are no updates"),
- reverting packages causing the downgrade of software,
- mix and match, where old signed repository snapshots are handed out.

An attacker might alternatively change dependencies to cause malicious effects. This involves changing the dependencies of a popular package to:

- provide everything,

- depend on everything,
- have unsatisfiable dependencies,
- include extraneous dependencies.

For dependency attack prevention, one has to consider that the manifest is a "database descriptor", which tells the client what are the latest versions of packages and where to find them. Since this can be significant in size, design decisions for download sizes and such have to be taken into consideration when designing systems safe against these kinds of attacks.

As a solution to some of these problems, Cappos presented the stork package manager which was specifically designed for the cloud. The result of this project is advice for where and how to sign and hash to make package managers more secure. Furthermore, the package manager TUF is presenting a more resilient solution to secure update handling by distributing on delegating the burden of signing into roles. In this sense, the delegation, separation, and explicit/implicit revocation of roles is the security base of TUF. However, there are sill issues with TUF remaining. This includes project registration with online or offline keys, ambiguous delegation, and discontinued projects.

Question Why do we not take a signature over the whole ZIP file instead?
Answer This depends on the implementation and is a design decision with implications on the whole repository and metadata structure, not just for one package. Some managers put the signature in the archive as metadata manifest file, but the manifest file can also be turned into a bomb.

Question Are there no signatures on the dependencies?
Answer This depends on the package manager. Some have the dependency files they use separate and then this is not checked for signature or hash. This is distributed separately.

6 Secure development and update of connected cars

The second session of Justin Cappos was on the application of secure software updates to connected cars. First, he started by introducing a typical setting of sensor and actor nodes in a car, e.g., engine control unit, transmission control unit, anti-blocking break system sensors, etc. These units are commonly called electronic control units or short, ECUs. Later, he discussed options for secure updates in the setting of these constrained nodes and possible attacks.

The entities involved in the software updates of connected cars span from an entity at the OEM and many entities in the car. At the OEM, i.e., car producer or vendor, we find the repository manager, which is handing out updates to the cars. In the car, we have a primary ECU which provides the communication

link to repository manager. Furthermore there are many secondary ECUs, which receive the updates by the primary ECU.

How can we ensure secure updates of secondary ECUs in case the primary ECU is compromised? One option is to append hashes and signatures to the updates, which have to be checked by the respective secondary ECU prior to an update. The same goes for requesting an update from the repository manager, since the primary ECU could change the request. In this case signing device type and version information for the request would help, e.g., by using an HMAC.

On the other hand, some nodes might be relatively resource-constrained, restricting them to a partial verification, which does not require the update to be downloaded completely. Therefore, the device could be implemented in a way to just require a minimal set of resources for operating during the update, i.e., limb mode. Then, the update could progress and replace the software in blocks one by one. For these blocks there have to be enough resources to store one temporarily, check its signature, and overwrite the existing software to continue with the next block. An A & B memory for the limb mode software would allow most efficient use of this temporary memory to reduce the overall memory requirement even further. As a side note, a merkle tree is not a useful structure for storing the block hashes in this case. A simple list of blocks is a good choice here.

An issue could be created by the constrained devices lacking a wall clock but the solution is simple. The devices can obtain the time by sending a nonce to a server, e.g., the repository manager, which in turn sends back the time together with the nonce signed. In this way, they cannot be tricked into believing there is no new update.

This renders us with attacks on the software distribution and key corruption side as discussed in the section before. We observe many exploitable issues in development and build systems. There can be backdoors placed in a project's source control, in packaged GCC distributions, or other development tools. These would be used for bootstrapping to compile GCC itself and thereby possibly adding the backdoor which is then in the following project compilation steps added to the final binary. To this end, verifiable compilers help by proofing that the produced code is related to the source code. The usage of reproducible compilers are an alternative. They always produce the same output and a hash attached to the source code could make behavior preserving checkable in this way.

Because of its interest for attackers, software updates (especially for resource-constrained devices) could be an application for creating provably secure software as part of my PhD project.

Question How about real-time Ethernet in cars? Would this imply less involving solutions on the software updating procedures since this is not a broadcast network?

Answer Yes, possibly. Ethernet is more resilient and there is also secure CAN, which mitigates some problems but it would not solve all of them. Cappos believes some of the problems will always be there independently of the

network.

7 Code execution protection using SGX

Mark Ryan presented in the first part of his session SGX's fundamentals and how they are intended to protect code execution. SGX is short for Intel's secure guard extensions and these introduce and use the notion of enclaves to isolate code and data from the operating system (OS) and other applications. Additionally, local and remote attestation are supported. In the second part, Ryan presented an interesting potential application for security-privacy-tradeoff.

As a hardware security anchor, SGX is not the first and therefore has to be compared to TPM and ARM TrustZone. The TPM API allows to create keys, for which the secret part never leaves the TPM chip. Afterwards, the secret key is accessed by so called authdata, which is like a passphrase, and/or the measurement of the code running. With ARM TrustZone, the hardware enforces access control between two modes of operation orthogonal to the user-kernel-isolation, i.e., the so called normal and secure world.

SGX protects enclaves from attacks by a virtual machine monitor (VMM) and/or OS while their code runs in ring 3, which is user mode. However, certain attacks are not addressed by SGX, which are side channel (cache and page access patterns) and hardware (chip decapsulating and trojan hardware in the supply chain). While SGX is good for cloud, where machine interaction is prevalent, it is not good alone for platform I/O, which means in interactions with a user (e.g., a password manager). The existing alternatives to SGX in the cloud scenario are all no generally applicable solutions. Ryan's big criticism on SGX is that it is not the work of a consortium but rather just a single company, i.e., Intel.

Following the API, SGX enclaves are created dynamically and it has to be specified during creation how they are identified. The caller can choose to identify an enclave by its measurement (`MRENCLAVE`, "measurement of enclave", strict) or by author information (`MRSIGNER`, includes software vendor product and version id, this allows software updates).

SGX implementations themselves are split into two parts. On one side, there are processor instructions, which are implemented in microcode and hardware. On the other side, there are Intel infrastructure enclaves for launching, provisioning, and supporting enclave operations like attestation and updates. The secret values in the platform are Intel's master derivation key, which is known to Intel, the seal secret `SEAL_KEY`, and `OWNER_EPOCH`.

A standard task is turning an existing application into an SGX secured application. Ryan notes that it would be more secure, as in better control, if you do not make system calls from the enclave but just return data to the application, which in turn makes the required system calls. The shorter and smaller the enclave operations the better. In this way an enclave exposes less surface to attackers in several senses, e.g., execution time and API. In practical terms, this means identifying the very necessarily secure parts of an application and implementing exactly only this in an enclave.

A typical use case of SGX is fast encryption on CPU. Therefore, a key can be inserted into an enclave and then be bound to only it by simply using SGX's sealing. For this purpose, SGX key derivation can be used as well. This is initiated by `EGETKEY` and depending on whether the caller supplies the `SEAL_FUSES` flag or not, the key can be reproduced on another platform or not. The `OWNER_EPOCH` hardware field can be used as well, which represents ownership and rejecting future owners access to keys derived in the present. Therefore, a user has to reset this flag before selling or giving a CPU or computer to a new owner. Furthermore, minor fields like product and version id can be taken into account as well. It is worth to note that the key for sealing a blob is always derived from the `SEAL_FUSES`. This blob can then be written to disk or elsewhere. Later it will be possible to unseal the blob after reboot by deriving the same key.

As an interesting application, Ryan motivates for accountable decryption using an escrow entity. The example for this is a teenager who want to going out but his parents normally do not need to know the place. However if the teenager does not come back, the parents should be able to find out to ensure safety of their child. A solution for this can be an envelope with an enclosed sheet, where the place is written down. This can be considered as a sufficient privacy and security trade-off. Once the parents open the envelope without any arguable reason, the child would be able to see this. Further applications could be government investigations concerning their citizens, corporate mail concerning their employees, or phone location sent encrypted on vendor servers. For the latter, an alternative solution with a passphrase puts the burden on the user in terms of a password choice.

Assuming that there is no mathematical crypto solution for this problem, since data can be arbitrarily copied, what is required for such a hardware? One idea is a blockchain on a crypto device to record key usages. Furthermore, a merkle tree could be used for efficient maintaining of hashes. Verifying hash inclusion is also efficient, i.e., the proofs that two root hashes correspond to append-only trees are relatively short.

In order to prevent fork attacks to the blockchain log, cryptographic beacons could be incorporated. This prevents the snoopy entity from showing an old clean log to the user. Ryan proposes to force the snoopy entity by the protocol to regularly invoke signing operations of the blockchain head and an unpredictable but user verifiable value. The advantage of this is that the user does not need to make the snoopy entity aware of the fact that privacy is a concern.

The first part of this session is reviewing a hardware security extension that I could use in my research. In this case, I could create a model of SGX first or alternatively use existing formal models of SGX in order to proof isolation in a certain piece of software. This could be either an application like it is described in the second part of this session or possibly a more generic isolation method to provably remove side-channels or to implement provable application isolation.

Question Is there a separate memory for enclaves?
Answer No, but the data is encrypted as it moves from processor cache to

main memory and on the way back it is decrypted again.

Question Is there a slowdown cause by the encryption and decryption? Furthermore, how are the monotonic timestamps implemented by service enclave?
Answer There is a little overhead with the encryption between cache and memory, as a random key is determined at boot-up. The timestamps are maintained on flash memory on the CPU chip and the implementation tries to make little writes to balance lifetime and timestamp period.

8 Reliable side-channel security

In the last (but not least) session, Alexandra Weber presented three approaches to reliable side-channel security:

- information theory to quantify information gain,
- type systems to track processor cycle timing leakage statically, and
- testing to determine distinguishability.

The information theoretic approach helps to quantify how much information is gained from a respective side channel, e.g., a timing channel. Weber's approach uses over-approximation by abstract interpretation. The choice of the abstraction mapping and the result relation is crucial in this approach.

To exemplify the type system approach, Weber presented a type system for AVR microcontrollers, i.e. SCF^{AVR}. This type system is augmented with a clock cycle counter, which has to match at the merging point of branches in order to result in a correct typing. In the end, she showed an evaluation with the μNaCl library to prove parts of it timing-leakage free.

The testing approach was implemented and the results were produced with Java. For this purpose, a quantity representing the side-channel has to be determines, i.e., execution time. To this end, Weber presented a histogram of execution times of many encryption runs, which have been collected by encrypting with two different keys. With a timing like, e.g., the naive implementation, the two resulting distributions are distinguishable.

By applying simple techniques for branch time leakage elimination, we can make them indistinguishable. One of these techniques are bitslicing as Schwabe added in a comment. He and his colleagues implemented AES with timing-attack resistance and report on it in their CHES 2009 article "Faster and Timing-Attack Resistant AES-GCM". They implemented this on x86 with SSE extensions by using bitwise operations like AND and OR. The result is bitwise parallel and similar to an implementation in hardware.

Many of the topics of this session, I have already encountered during my previous PhD studies. We are working with system software security, where provable side-channel elimination together with reasonable machine modeling are possible results.

Question In the type system, why is there a number associated in the conclusion of the branch type inference rule and what does it mean?
Answer This number represents the processor cycles that have been consumed in this place of the program. We want to correctly type programs iff their execution takes constant time in all cases. To this end, we have to keep track of and embed the processor cycles into the type system.

9 Conclusion

While the application areas of resource constrained devices are quite broad, some of them are quite delicate as well. Without exaggeration, lives are at risk and practices need to be changed. Cars should definitely not be able to be hacked and controlled wirelessly and freely at least. To this end, software updating has shown to be crucial and very vulnerable in one of the sessions. At the same time, my research subject the critical infrastructures need proper protection as well and similar considerations apply here as well. Intruders should be forced, at least, to break cryptographic protocols or similar.

The challenges for secure implementations of cryptographic functions on small devices are diverse, but yet they have to be taken. If we are dealing with devices that are out in the field, attackers have more opportunities for cryptoanalysis and in such cases a solid side-channel resilience has to be in place. Alternatively, the system can be build differently such that a single device compromise has not so many implications or similar.

Furthermore, a basic understanding of the attack vectors and a tool set to prove security are essential for this task. Formal verification, for example, is a basic tool to prove correctness of certain properties. Side channels analysis, on the other hand, is required to evaluate attacks beyond the models typically used in a formal verification exercise. This involved new challenges on formal verification, which is very much related to my research project.

Finally, I want to thank Hanno Langweg and Urszula Nowostawska for organizing the summer school, all the participants for creating an open atmosphere, and also the speakers in the order of their sessions. Thank you Paris Kitsos, Lejla Batina, Peter Schwabe, Justin Cappos, Mark Ryan, and Alexandra Weber.