

Combating Side-Channel Attacks Using Key Management

Donggang Liu and Qi Dong
iSec Laboratory, Department of Computer Science and Engineering
The University of Texas at Arlington

Abstract

Embedded devices are widely used in military and civilian operations. They are often unattended, publicly accessible, and thus vulnerable to physical capture. Tamper-resistant modules are popular for protecting sensitive data such as cryptographic keys in these devices. However, recent studies have shown that adversaries can effectively extract the sensitive data from tamper-resistant modules by launching semi-invasive side-channel attacks such as power analysis and laser scanning. This paper proposes an effective key management scheme to harden embedded devices against side-channel attacks. This technique leverages the bandwidth limitation of side channels and employs an effective updating mechanism to prevent the keying materials from being exposed. This technique forces attackers to launch much more expensive and invasive attacks to tamper embedded devices and also has the potential of defeating unknown semi-invasive side-channel attacks.

1. Introduction

Wireless embedded devices have been extensively used in civilian and military applications. They range from small systems like medical sensors and automobile engine controllers to large infrastructures like traffic lights systems, security monitoring systems, and tactical wireless networks. Security has been recognized as an integral part of these embedded systems, especially when they are physically accessible to unsupervised people.

Embedded devices are often deployed unattended and can be easily captured by adversaries. Once they are captured, the adversary can immediately extract sensitive information such as keys from these devices and then arbitrarily manipulate them or even replace them with malicious devices. For example, after an adversary captured a video sensor in a security monitoring system and extracted the secret key inside, he can either inject malicious codes or replace it with a more powerful malicious device to gain more control over the system. As a result, the adversary can forge/modify event reports to subvert the intended purpose (i.e., security monitoring) of the system or leak sensitive information such as the locations of certain objects (e.g., persons). It is thus important to guarantee the integrity of embedded devices even if they are physically captured by adversaries.

Typical techniques for ensuring the integrity of embedded devices include *tamper-resistant hardware* and *code attes-*

tation. Tamper-resistant hardware use an on-chip sensory circuit to sense the physical tampering and erase secrets when detecting any tampering activity. It is well known that with sufficient investment and time, adversaries with access to advanced semi-conductor equipment can directly observe or even manipulate the chip components to retrieve the keying materials. Fortunately, an adversary with such capability will need to have an advanced semi-conductor laboratory that may cost millions of dollars. As a result, tamper-resistant hardware modules are still quite useful in many scenarios. However, a number of recent studies have shown that there are many low-cost, *side-channel* attacks such as power analysis or laser scan to extract keying materials from the tamper-resistant hardware [2], [13].

Several hardware solutions are suggested to protect embedded devices against semi-invasive side-channel attacks [8], [10], [11], [14], [18]. These solutions use hardware components to reduce symptoms that may leak sensitive information. However, these solutions are designed to address some specific classes of side-channel attacks. As technology advances, attackers may discover new side channels to read data from tamper-resistant modules. Once this happens, we may need to upgrade the existing hardware modules, which can be prohibitively expensive for a widely-deployed system. Additionally, most hardware solutions are not cost-effective. Thus, it is also important to develop *low-cost* counter-measures that can effectively withstand *unknown* side-channel attacks.

In addition to tamper-resistant hardware, code attestation techniques are also quite popular for checking the integrity of the software on embedded devices [15], [16]. Some of them use hardware support such as TPM (Trusted Platform Module), while some others use purely software-based approaches. However, these solutions have many limitations in practice. In particular, TPM are actually not tamper resistant. Software-based approaches often assume that the attested device has the same hardware architecture as the original untampered one. However, the assumption of the same hardware architecture may not hold at all. In particular, the adversary can extract the keying materials (through side-channel attacks) as well as the original code from the embedded device and replace the device with a more powerful malicious device that emulates the victim device. This makes software-based approaches mostly ineffective.

In this paper, we focus on techniques to ensure that no adversaries can compromise an embedded system via side-

channel tampering. Indeed, our proposed technique allows a *benign embedded device to verify whether another embedded device has been tampered or not*. This is achieved by proposing a novel key management technique to leverage the bandwidth limitation of side channels. That is, with limited access to advanced equipment, it is extremely hard for a side-channel attacker to retrieve a large portion of the protected secret from a tamper-resistant module within a short period of time. In most cases, the attacker can only read out a small portion (e.g, a few bits) of the actual secret in each tampering activity. The information recovered by the attacker consists of the values from many tampering activities conducted at *different times*.

Based on the above observation, this paper proposes an efficient and effective technique to harden embedded devices against semi-invasive side-channel attacks. The contributions of this paper are as follows. First, we present an efficient updating scheme to refresh the secret at every embedded device before it is fully exposed. This technique ensures that capturing an embedded device will not allow an attacker to recover the entire secret, making it possible to check whether an embedded device has been tampered. This updating mechanism can also effectively force an adversary to conduct a much more expensive and invasive attack to tamper an embedded device. Since most side-channel attacks are limited in their tampering bandwidth, this technique can also defeat *unknown* side-channel attacks. Second, we also show how to use these frequently-updated secrets to support security protocols such as key establishment and detect captured devices.

The remainder of the paper is organized as follows. The next section reviews recent studies on tamper-resistant hardware as well as code attestation techniques. Section 3 discusses the adversary model. Section 4 describes our key management technique for hardening embedded devices against side-channel attacks. Section 5 evaluates the proposed technique. Section 6 concludes this paper and points out some interesting future directions.

2. Related Work

Tamper-resistant modules are commonly used to protect critical keying materials when the devices fall into wrong hands. However, it is well believed that given sufficient investment, an adversary can always recover the hidden data from any tamper-resistant hardware. Nevertheless, when an adversary has limited funds and access to advanced equipment, tamper-resistant modules such as Smartcards are still very useful. Unfortunately, several recent research studies have shown that a number of low-cost attacks, called *side-channel attacks*, can be quite effective in compromising the security of most tamper-resistant modules [2], [13]. These attacks allow any individual with limited access to advanced equipment to do the tampering effectively. Typical side-channel attacks include power and timing analysis [4]–[6], [8], [9], [11], [17], fault injection attacks [3], [7], laser scan [14], and electromagnetic analysis [14].

Many hardware-based solutions have been proposed recently to deal with the above semi-invasive side-channel attacks [8], [10], [11], [14], [18]. Randomization is suggested to address the side-channel attacks that require precise knowledge about the code execution [10]. Data masking and noise introduction are used to counter power analysis [8], [11]. Electromagnetic analysis can be defeated by applying aggressive shielding or breaking the locality of the chip layout [18]. Fault injection attacks can be mitigated by using sensors that monitor environmental changes [14]. In this paper, we will propose an efficient and low-cost key management technique for an additional layer of protection in addition to the hardware-based solutions. This scheme also has the potential of defeating those unknown semi-invasive side-channel attacks.

Another option to ensure the integrity of embedded systems is *code attestation*. Examples include SWATT [15] and BIND [16]. SWATT is a purely software-based approach where a verifier asks for the hash of the content of randomly selected code blocks. To provide a correct answer, the responder has to keep the original copy of code, which turns out to be quite difficult for a malicious device that has the same hardware. However, this assumption is unrealistic in situations where the adversary can replace embedded devices with more powerful ones. BIND [16] is based on a TPM chip to check the integrity of the code and static data. Due to the side-channel attacks, this only gives a partial solution. Our scheme leverages a technological limitation of the attacker to enhance the resilience of embedded devices. It is purely software-based and can be combined with previous solutions to provide better security.

3. Problem Statement

We consider a network of embedded devices in systems like wireless mesh networks [1]. We assume that an embedded device can be physically accessed by unsupervised adversaries for a certain period of time. During this period of time, the adversary can access most parts of the device and manipulate them physically to some extent. For example, he can probe and monitor the communication link between the tamper-resistant module and the external memory module. However, we do not consider those *invasive attacks* where the attacker has access to advanced semi-conductor equipment to physically open the tamper-resistant module and make direct contact with the circuit inside. Instead, *it is considered as a success if our approach forces the attacker to launch a much more expensive and invasive attack*.

This paper focuses on side-channel attacks. Specifically, we assume that attackers can read out the protected content of the tamper-resistant module via some side channels discussed before. We found that a common feature of side-channel attacks is their *limited bandwidth*. For example, in order to launch power analysis attacks, one has to observe the power consumption of the tamper-resistant module for a significantly long time to gather sufficient information to conduct statistical analysis.

In this paper, we model a side-channel tampering attack as a serial of *tampering experiments* conducted by the adversary. Each experiment will take at least t minutes and can recover up to b bits of the protected data content (cryptographic keys) from the tamper-resistant module. We simply call such adversary as the (b, t) -capable adversary (or simply (b, t) -adversary). The actual values of b and t here depend on how much effort the attacker is willing to invest to compromise the secret information on the tamper-resistant module. Determining such parameters usually requires a thorough investigation of the adversary model in specific applications. We will not include such investigation in this work since it is beyond the scope of the paper. Therefore, our focus is to ensure that leaking b bits of secret information every t minutes will not give the adversary any advantage to recover the entire secret.

We assume that the cryptographic algorithm in a tamper-resistant module can always run without interruption even if the corresponding embedded device has already been captured by adversaries. This assumption can be made true in most cases. For example, the power and timing analysis require that the algorithm in a tamper-resistant module run correctly during the tampering to collect sufficient statistical information. However, we also note that in some side-channel attacks such as laser beam scan, an attacker can stop the power supply or clock to make any software-based approach ineffective. However, these problems can be handled by keeping secret keys in volatile memory so that they will disappear in case of power loss and having an internal clock that can hardly be disabled. Thus, in this paper, we will simply assume that a side-channel attacker cannot disable the power supply or clock without destroying the secret physically.

4. Proposed Approach

In this section, we introduce our key management technique for defeating side-channel attacks. The main idea is to let every embedded device update the secret in its tamper-resistant module before the secret is fully exposed. Hence, the partial secret recovered earlier will be of little use to the adversary during the tampering. However, updating keying materials frequently makes it particularly challenging to build security protocols based on these ever-changing keying materials. It is thus imperative to design an updating scheme that provides support for various security protocols.

With this in mind, we notice that the most common use of the keying materials in an embedded device is to setup session keys to protect the communication with other devices or central servers in an embedded network. As a result, in this paper, we will also propose techniques for the establishment of session keys between devices using those frequently-updated keying materials. Thus, two objectives of our work are to make the scheme *hard to tamper* and, at the same time, *easy to use* for session key establishment. It is certainly possible that the keying materials are used for different purposes other than session key establishment. We

<p>Input: Address of array A_u Result: Updated array A_u $tmpstr = A_u;$ $A_u[0] = H(tmpstr 0);$ $A_u[1] = H(tmpstr 1);$ \vdots $A_u[b'] = H(tmpstr b');$ $erase\ tmpstr;$</p>
--

Figure 1. b -Resistant Function RF_b

will not cover those uses in this paper and instead consider them as possible future directions.

We will start our discussion with a baseline approach that prevents the adversary from learning the *master secret (key)* of a captured embedded device through side-channel tampering. We will then discuss how to use the master secrets to establish session keys for secure communication between embedded devices and how to detect those devices that have been captured and tampered by adversaries.

4.1. The Baseline Approach

Our baseline approach is based on two intuitive ideas. The first idea is to make the master secret large enough so that a (b, t) -adversary can only have a partial view of it in every tampering experiment. The second idea is to update the master secret stored in the tamper-resistant module periodically and fast enough such that the collection of the partial secrets discovered in different experiments won't help the adversary to recover the entire master secret.

Our approach requires that the master secret (key) of every embedded device be uniquely and randomly selected. This key is only known by a central server and the device itself. This requirement is made because of the possibility of an attacker tampering many devices in parallel. Indeed, if the same master key is used by many devices, it is possible for the attacker to tamper these devices at the same time and pull the partial secrets recovered from different devices together to infer the master secret.

Our protocol will use a secure one-way hash function H , which takes an input with arbitrary length and outputs a hash image with length of m bits. This function has the one-way property as well as the weak collision-free and strong collision-free properties. For simplicity, we assume that the length of a cryptographic key is also m bits. To deal with a (b, t) -adversary, we simply have every embedded device run a b -resistant function every t minutes. The pseudo code of such function is given in Algorithm 1. For simplicity, we assume that $b = m \times b'$. The master secret will be pre-loaded to the tamper-resistant module of every embedded device u by a trusted server during the initialization and saved in the array A_u , which can hold $(b' + 1)$ hash images (i.e., $b + m$ bits in total). We assume that this initialization phase is done without the presence of adversaries.

Let $RF_b(\cdot)$ denote the above function. Note that every execution of function RF_b updates the content of array A_u .

Over time, we will have a chain of master secrets that are linked by function RF_b . For convenience, we let $A_{u,i}$ denote the master secret of device u after the i -th update using function RF_b .

It is possible that the execution of function RF_b in an embedded device is interrupted by, for example, power failure. The master secret in this device may be partially old and partially new and will be likely unrecoverable. Once this happens, we will have to re-initialize the device. Since re-initialization can be expensive in some applications, we propose to install batteries on every embedded device so that short-term power failure will not affect our approach. Indeed, installing batteries on embedded devices are quite popular in current practice. There might be other effective alternatives that can address the interruption of function RF_b . However, we consider them as one possible future direction.

Overheads: The baseline approach only incurs small overheads. First, there is no additional communication overhead needed for updating the master secret. Every embedded device can update its master secret independently. Second, the additional storage overhead mainly comes from the space needed for saving the master secret, which only occupies $b + m$ bits of space in the tamper-resistant module. Finally, every embedded device only needs to do $(1 + \frac{b}{m})$ hash operations every t minutes. In typical scenarios, this will not be a big problem for embedded devices.

Parameters b and t : Two critical parameters in our baseline approach are b and t . Clearly, a larger b leads to a larger storage overhead for each embedded device and more hash operations involved in each round of the b -resistant function. We thus prefer a small b for high efficiency. However, for security purposes, we need a larger memory space to deal with more powerful adversaries that can extract more information from embedded devices in each tamper experiment. In this sense, we prefer a large b for high resilience against side-channel tampering. Hence, a trade-off has to be made between security and efficiency in selecting b .

Since each embedded device needs to perform the b -resistant function every t minutes. A small t will certainly lead to high computation overhead for each embedded device. However, a small t will also make it more difficult for attackers to tamper the device via side-channel attacks since the attacker has to finish the tamper experiment in no more than t minutes. As a result, there is also a trade-off between security and computation overhead in selecting t .

As we mentioned, making trade-offs in determining b and t requires a thorough investigation on the adversary's capability in a specific application. One way is to figure out how much effort an attacker is willing to invest to tamper embedded devices. However, it is difficult to quantify the attacker's investment. We thus propose to configure b and t using a "best effort" strategy based on the computation and storage capabilities of embedded devices. In other words, we want to find out the largest b and smallest t that will not impact the normal operation of embedded devices.

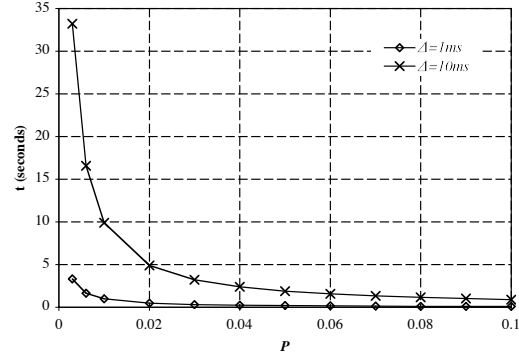


Figure 2. Configuration of t ($b = 9 \times m$)

We propose to set b as large as possible in this paper as long as the storage space allows and the b -resistant function RF_b can be efficiently done. This is not a problem for a given hardware platform. To configure t , we will need to know how much computation resource the application is willing to invest. For simplicity, the fraction of CPU time is used to quantify the computation resource. Let P be the fraction of CPU time that can be allocated to perform our proposed updating method without affecting the normal operation of the embedded device. Determining a proper P is generally not a problem for a given application. Let Δ be the time (in seconds) needed to perform a single hash function H . Note that function RF_b requires $(1 + \frac{b}{m})$ hash operations every t minutes. We thus have the following equation:

$$\frac{(1 + \frac{b}{m}) \times \Delta}{60 \times t + (1 + \frac{b}{m}) \times \Delta} \leq P.$$

Hence, $t \geq \frac{(1-P) \times (b+m) \times \Delta}{60 \times m \times P}$. This tells us that we can simply set $t = \frac{(1-P) \times (b+m) \times \Delta}{60 \times m \times P}$ to achieve high security guarantee and at the same time meet other constraints. This actually provides a guideline to configure parameters b and t properly. For example, assume that it takes about $1ms$ to perform a hash function H . If every execution of the b -resistant function RF_b has to be done in $10ms$, we can set b as $9 \times m$. Figure 2 shows the configuration of parameter t under different situations, assuming $b = 9 \times m$. The size of the master secret is $10 \times m$ bits long. Suppose 1% of CPU time can be allocated to our updating method ($P = 0.01$). According to this figure, we can simply set $t = 1$ second. After configuring all the above parameters, we can see that the integrity of embedded devices can be ensured as long as the adversary cannot recover more than $9 \times m$ bits of secret information in one second.

Security: Our security analysis focuses on how hard it is for a (b, t) -adversary to recover the entire master secret from a device running the b -resistant function every t minutes.

The security of our approach can be easily explained as follows. Since the master secret is $(b + m)$ bits long and every single tamper experiment gives at most b bits of secret information to the adversary, we know that for

any i , there are at least m bits of $A_{u,i}$ (the master secret after the i -th update) that cannot be directly tampered by a (b, t) -adversary. Since the entire secret $A_{u,i}$ is used to generate every hash image of $A_{u,i+1}$ (the master secret after the $(i + 1)$ -th update) using the one-way function H , it is computationally infeasible for any (b, t) -adversary to recover the entire $A_{u,i+1}$ even if the adversary has extracted b bits of $A_{u,i+1}$ using side-channel tampering attacks. Hence, for any i , at least m bits of the master secret in a captured embedded device are unknown to the (b, t) -adversary. This tells us that as long as function H is a secure one-way function and m is large enough, the security of the master secrets in those captured embedded devices can be guaranteed.

4.2. Session Key Establishment

Our baseline approach can effectively hide the master secrets from side-channel attackers. In addition to this, we have to make sure that the master secrets can be easily used for security purpose. For example, in wireless mesh networks [1], the embedded devices will be organized as a multi-hop wireless network to provide network access to users. These devices will need to be able to talk to each other securely in various protocols such as routing. However, the master secrets cannot be directly used for security needs such as encryption and authentication since they are randomly picked and change from time to time. Next, we will focus on how to setup session keys between embedded devices using their master secrets.

Our proposed solution will use a Key Distribution Center (KDC), i.e., the trusted server used to pre-distribute the initial master secrets. An embedded device will need to contact the trusted server to setup a session key with another device. We assume that the trusted server is installed in a well-protected place; it will never get captured during the normal operation of embedded networks. The main issue now is *to establish a session key between a given embedded device and the trusted server*. Once every embedded device shares a unique secret key with the trusted server, there are many KDC-based techniques available for us to implement session key establishment [12]. Note that although the trusted server knows the initial master secret of every embedded device in the system, this secret cannot be directly used as the session key between the device and the trusted server since it changes quickly during the network operations. We address such key establishment problem as follows.

Let $A_{u,j}$ be the current master secret in the embedded device u . When device u needs to setup a session key with the trusted server, it will directly compute the hash image $K_{u,j} = H(A_{u,j} || (b' + 1))$ as the session key. Meanwhile, it will send a message to the trusted server for key establishment. This message will include the value j and will be protected by the session key $K_{u,j}$. When the trusted server receives the message, it can immediately compute the current master secret $A_{u,j}$ at device u based on the initial secret $A_{u,0}$ and j . With $A_{u,j}$, it can then compute the same key $K_{u,j} = H(A_{u,j} || (b' + 1))$ as device u ; this

key can then be used for authenticating the message. Once the authentication succeeds, the device and the trusted server can further exchange messages to agree on the same key. The communication between device u and the server S is illustrated in the following:

$$\begin{array}{l} u \rightarrow S : \{ \text{Nonce}, u, j, H(\text{Nonce} || u || j || K_{u,j}) \} \\ S \rightarrow u : \{ \text{Nonce} + 1, H(\text{Nonce} + 1 || K_{u,j}) \} \end{array}$$

The session key established in the above way will only be used for a short period of time for security purpose. Its actual lifetime depends on the application. When the current session key expires, a new session key can be established in the same way. Since the trusted server now shares a unique key with every embedded device, it is quite trivial to establish session keys between embedded devices using KDC-based approaches like the one in [12].

Overheads: Our session key establishment protocol requires a KDC-based protocol for establishing keys between embedded devices. There are many practical KDC-based approaches available for key establishment [12]. Hence, we believe that the storage, computation, and communication overheads involved in such KDC-based key establishment protocol will not be a big problem for embedded devices. In addition to these overheads, we also see that an embedded device only needs m bits of storage space, computes one hash operation, and delivers one message to the trusted server to establish a session key with the trusted server. We thus conclude that our session key establishment protocol is practical for embedded devices.

Security: We note the the session key established in our protocol will not change as frequently as the master key. This clearly allows a (b, t) -adversary to extract the session key from the module with one side-channel tamper experiment. In the following, we analyze how this will affect the security of the embedded systems.

There are two kinds of session keys, *class-one session keys* and *class-two session keys*. The class-one session keys are those shared with the trusted server, and the class-two session keys are those shared between embedded devices. According to the protocol, we note that the class-one session keys are generated from the master secret using the one-way hash function H . Hence, we can clearly see that no matter how many class-one session keys are extracted by an adversary, the security of the master secret will not be affected, assuming that the underlying cryptographic algorithms such as encryption and hash function H are secure, and m is large enough. In addition, we also note that the class-two session keys are established through a KDC-based key establishment protocol using the class-one session keys. Therefore, no matter how many class-two session keys are compromised, the security of the master secrets as well as the class-one session keys will not be affected.

The above analysis shows that a (b, t) -adversary can not recover the master secret. However, we also note that it is still possible for the adversary to impersonate captured devices since (1) all communications are protected

by session keys, and (2) the session keys can always be extracted directly by adversaries. In other words, a (b, t) -adversary can always conduct side-channel attacks to extract every session key established during the lifetime of an embedded device and continuously impersonate this device. Such impersonation allows the adversary to inject bogus messages in the communication between the captured device and any other benign device. As long as the attacker is willing to pay such cost and effort, this attack will be a great security threat to an embedded system. Next, we will present an efficient technique to deal with this problem.

4.3. Detecting Captured Devices

As we mentioned, the session key will be used for a certain period of time for secure communication. When the adversary captures an embedded device, it can directly extract the (m -bit long) session key stored in the memory. This allows the adversary to read the communication and impersonate this device. We must recognize that when the attacker only passively reads the communication, it will be extremely difficult, if not impossible, for us to detect such behavior since there is no clear evidence for such misbehavior. Fortunately, such passive eavesdropping will not generate much damage to the system. However, impersonating an embedded device and interacting with other devices actively can easily mislead or subvert the intended purpose of the application, generating significant damage to the system. Therefore, the main effort in this section is to detect such impersonation attacks.

Although the adversary can always know the session key by launching side-channel attacks continuously, he still don't know the master secret. In addition, the adversary cannot change the code running in the tamper-resistant module without destroying the master key. As a result, when an adversary injects malicious messages from a captured device u to a benign device v , the communication history from u to v seen by the two tamper-resistant modules on the two devices will be likely different from each other. Such inconsistency can be used for detecting captured devices. Certainly, the adversary may manipulate this and make the two tamper-resistant modules see the same set of messages. For example, he can simply feed the module on the captured device with malicious messages instead of injecting directly. However, doing this will significantly limit the impact of impersonation attacks since the attacker is no longer able to inject arbitrary messages. For instance, when the tamper-resistant module on the embedded device has an internal logic (as well as a buffer to store intermediate results) to produce all control messages, the attacker can only inject malicious data (e.g., a file in the flash memory). Injecting malicious data can be handled by a proper integrity verification method. In this paper, we assume that *the adversary always want to directly inject messages without being noticed by the internal logic of a captured tamper-resistant module.*

Our main idea is to have the embedded device commit the history of its communication using its master key whenever it is asked to do so. Such commitment value will be generated using the past communication to the device that is requesting for commitment. The usage of the master secret will make it impossible for any (b, t) -adversary to forge a correct answer even if it has captured the device. The commitment will be checked against the communication history seen by the requesting device. If there is any inconsistency, the requesting device know that the session key has been compromised and the device in question has been captured. The detail of this approach is given below.

Let us consider two communicating embedded devices u and v . Every message transmitted between them will include a sequence number. A TCP like protocol will also be applied to achieve reliable communication. For the communication from device v to device u , u will maintain a variable $C_{v,u}$ as the summary of the communication history from v to itself. This variable is initially set to zero and updated according to the following simple rule.

- Let i be the sequence number of the last authenticated message from v . When u receives an authenticated message M from v with sequence number $i+1$, it will update $C_{v,u}$ by computing $C_{v,u} = H(C_{v,u} || M)$.

Similarly, the device v will also maintain a variable $C'_{v,u}$ as the summary of the communication history from itself to u . This value is set to zero initially and updated according to the following simple rule.

- Let i be the sequence number of the last acknowledged message from v to u . If v receives a new message from u confirming the receiving of its message M with sequence number $i+1$, v will update $C'_{v,u}$ by computing $C'_{v,u} = H(C'_{v,u} || M)$.

The above two updating rules will be affected by the channel loss and out-of-order messages. However, the channel loss can be addressed by standard fault tolerance methods such as re-transmission, and the out-of-order messages can be solved by asking the embedded device implement a sliding-window technique similar to the one in the TCP protocol.

So far we have discussed how to synchronize the communication history from v to u . The communication from u to v will be processed in the same way. In the end, each embedded device will maintain two variables, one for each direction of the connection. Next, we will discuss how to commit the communication history and detect captured devices.

Figure 3 illustrate the main process for detecting captured devices. Suppose that device u is trying to verify the communication history from v . Let i be the sequence number of the last message from v to u . Device u will send a request to v for the commitment of the past communication. Assume $A_{v,j}$ is the current master key at device v . Device v will send the following message to u :

$$v \rightarrow u : \{j, H(A_{v,j} || C'_{v,u})\}$$

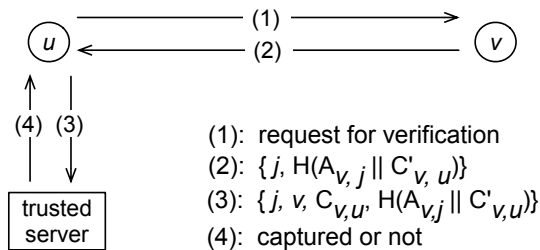


Figure 3. Detecting captured devices

This message will be authenticated by the session key between u and v . The purpose of using the session key is to ensure that nobody else can access the message. As a result, if there is anything wrong with this message, device u can immediately conclude that device v is captured since only device v knows this session key in addition to itself. The adversary who captured device v and extracted the session key can certainly read this message. However, he cannot modify the hash value since this requires the knowledge of the master secret $A_{v,j}$ at device v .

When device u receives this message from v , it will send a report $\{j, v, C_{v,u}, H(A_{v,j} || C'_{v,u})\}$ to the trusted server using a secure channel established by the session key between u and the trusted server. When the trusted server receives such message, it can easily compute $A_{v,j}$ based on j, v and A_v . It will then compute $H(A_{v,j} || C_{v,u})$. If this value equals the hash value included in the report, the communication history from v to u is authenticated; otherwise, the trusted server will notify device u to stop communicating with v . Such detection result will only be used by device u . The reason is to ensure that the captured device cannot send fake reports to mislead the detection.

Overheads: The proposed detection technique requires that every device keep two variables for every connection with another device. This is usually not a problem for embedded devices. As for the computation overhead, every outgoing message or authenticated incoming message will incur one additional hash operation. Each verification of communication history involves two hash functions at the device being verified and another two hash functions at the verifying device. This will not introduce much overhead for embedded devices. As for the communication overhead, there are only four messages involved in each verification. Overall, we can see that the proposed detection technique is practical for embedded devices.

Security: In our security analysis, we focus on the case where a benign embedded device u is trying to verify the past communication from a captured embedded device v . It is clear that any (b, t) -adversary has no way to recover the entire master secret at device v at any time. This indicates that the adversary cannot give a correct answer to u 's request for commitment. Additionally, if device u has ever received

a forged message from the adversary who impersonates v with the session key extracted from v 's memory, the variable $C_{v,u}$ maintained at u will be different from the one maintained at the tamper-resistant module of device v , i.e., $C'_{v,u}$. The trusted server will thus notice the difference and conclude that v is untrustworthy as long as the report about v from u reaches the trusted server. This indicates that the adversary has no way to inject forged messages directly into the communication between a captured device and a benign device.

The adversary who impersonates device v can certainly generate false reports about any other embedded device. The verification process at the trusted server will make a wrong decision even if the device in question is actually benign. However, we note that this will not impact our system since the result from the trusted server will only be used by the device who sends the report. Thus, a benign device will never get accused by any other benign device of being captured. Based on the above discussion, we can see that our detection technique can successfully detect if the adversary has captured and impersonated an embedded device.

5. Discussion

In this section, we will discuss how the proposed approach can be used to provide protection against various side-channel tampering attacks such as power and timing analysis, fault injection attack, laser scan and electromagnetic analysis.

Power and timing analysis: A successful power or timing analysis requires the adversary to collect a large amount of statistical information about the execution of a cryptographic algorithm using the *same* key. By applying our approach, i.e., running the b -resistant function every t minutes, we force the adversary to finish his power or timing analysis in t minutes and also recover more than b bits of information about the master secret. Achieving this can be extremely challenging for the adversary. In addition, we also note that a normal device will not invoke the cryptographic algorithm at a high frequency, while the attacker has to invoke the algorithm a large number of times to gather sufficient statistic information. Therefore, we can further enhance the security of the system by limiting the number of times the cryptographic algorithm can be invoked within t minutes. Doing this will not impact a normal system much but will make it particularly difficult for the adversary to gather sufficient information for analysis. Overall, we can see that our approach can significantly improve the security of embedded devices against power or timing analysis without requiring any hardware support.

Fault injection attacks: The fault injection attacks can physically impact the internal logic (e.g., the hash function H) in the chip without opening it. As a result, they do introduce unique challenges to the system that employs our approach since the chip may directly release the master secret if the adversary can successfully induce faults into its internal logic. Nevertheless, we found that our approach can

still work in general sense as long as the function H can limit the bandwidth of faulty instruction attacks. For example, the function H should not reveal much information about the master secret even if the attacker induces a few faulty instructions during its execution. Achieving this requires both the hardware support (e.g., making it more difficult to precisely induce faults at a given line of code) and software support (e.g., designing cryptographic functions that are more resistant to faulty instructions). These two directions are complementary to our work in this paper. They can be easily combined with our approach to achieve better security.

Laser scan and electromagnetic analysis: We note that these two classes of side-channel attacks are very limited in bandwidth. The reason is that the chip size of most tamper-resistant modules becomes smaller and smaller. It is thus particularly difficult for the attacker to precisely and quickly extract a large portion of the needed information from a tiny hardware module in a short period of time. This suggests that our proposed approach can work effectively against these attacks.

Remark: In general, we can see that if the attacker only has a bandwidth-limited side channel to tamper the secret stored in embedded devices, our proposed scheme can always work effectively against such attack. As a result, our technique has the potential of defeating unknown side-channel attacks. In addition, this paper indeed suggests an interesting direction for us to design tamper-resistant hardware. That is, instead of protecting every bit of the secret, we can simply make sure that the adversary's tampering bandwidth is limited. Intuitively, it will be much easier for hardware developers to limit the bandwidth of adversaries than to guarantee no leakage of secret information.

6. Conclusions and Future Work

In this paper, we present a key management scheme to protect embedded devices from side-channel tampering. We have shown that this approach can be quite effective in dealing with bandwidth-limited side-channel attacks. We pointed out that a very promising direction of developing tamper-resistant hardware is to limit the bandwidth of the attacker rather than to guarantee no leakage of any sensitive information, which may be extremely difficult to achieve in practice. Since our approach is developed using the bandwidth limitation, it can be used to defeat those *unknown*, bandwidth-limited side-channel attacks.

There are a number of interesting future research directions that we would like to work on. First, it is interesting to evaluate our scheme through real experiments to show how it improves the security of embedded devices. For example, we would like to show that the power or timing analysis is no longer able to figure out the master secrets in embedded devices. Second, we are also interested in efficient methods to limit the bandwidth of a side-channel attacker to further improve the security of embedded systems.

References

- [1] L. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: A survey. *ELSEVIER Computer Network*, 44:445–487, 2005.
- [2] R. Anderson and M. G. Kuhn. Tamper resistance - a cautionary note. In *Proceedings of 2nd USENIX Workshop on Electronic Commerce*, 1996.
- [3] D. Boneh, R. Demillo, and R. Lipton. On the importance of checking cryptographic protocols for faults. In *Proceedings of Eurocrypt*, 1997.
- [4] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium*, 2003.
- [5] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestre, J. J. Quisquater, and J. L. Willems. A practical implementation of the timing attack. In *Proceedings of the Third Working Conference on Smart Card Research and Advanced Applications*, 1998.
- [6] L. Goubin and J. Patarin. DES and differential power analysis. In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES)*, 1999.
- [7] S. Govindavajhala and A. W. Appel. Using memory errors to attack a virtual machine. In *IEEE Symposium on Security and Privacy (S&P)*, 2003.
- [8] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology – CRYPTO*, 1999.
- [9] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss and other systems. In *Advances in Cryptology – CRYPTO*, 1996.
- [10] O. Kommerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of USENIX Workshop on Smartcard Technology (Smartcard)*, 1999.
- [11] T.S. Messerges, E. A. Dabbish, and R. H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transaction on Computer*, 51:541–552, May 2002.
- [12] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks (MobiCom)*, July 2001.
- [13] S. Ravi, A. Raghunathan, and S. Chakradhar. Tamper resistance mechanisms for secure embedded systems. In *Proceedings of the 17th International Conference on VLSI Design (VLSID)*, 2004.
- [14] D. Samyde, S. Skorobogatov, R. Anderson, and J. Quisquater. On a new way to read data from memory. In *Proceedings of 1st International IEEE Security in Storage Workshop*, 2002.
- [15] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy (S&P)*, 2004.
- [16] E. Shi, A. Perrig, and L. V. Doorn. BIND: A fine-grained attestation service for secure distributed systems. In *IEEE Symposium on Security and Privacy (S&P)*, 2005.
- [17] S. Skorobogatov. Optically enhanced position-locked power analysis. In *Proceedings of Cryptographic Hardware and Embedded Systems Workshop (CHES)*, 2006.
- [18] D. Wong and A. Chan. Efficient and mutually authenticated key exchange for low power computing devices. In *Proceedings of ASIA CRYPT*, Dec 2001.