Weak-Unforgeable Tags for Secure Supply Chain Management

Marten van Dijk, Chenglu Jin, Hoda Maleki,
Phuong Ha Nguyen, and Reza Rahaeimehr

University of Connecticut
{marten.van_dijk, chenglu.jin, hoda.maleki,
phuong_ha.nguyen, reza.rahaeimehr}@uconn.edu

Abstract. Given the value of imported counterfeit and pirated goods, the need for secure supply chain management is pertinent. Maleki et al. (HOST 2017) propose a new management scheme based on RFID tags (with 2-3K bits NVM) which, if compared to other schemes, is competitive on several performance and security metrics. Its main idea is to have each RFID tag stores its reader events in its own NVM while moving through the supply chain. In order to bind a tag's identity to each event such that an adversary is not able to impersonate the tag's identity on another duplicate tag, a function with a weak form of unforgeability is needed. In this paper, we formally define this security property, present three constructions (MULTIPLY-ADD, ADD-XOR, and S-Box-CBC) having this security property, and show how to bound the probability of successful impersonation in concrete parameter settings. Finally, we compare our constructions with the light-weight hash function PHOTON used by Maleki et al. in terms of security and circuit area needed. We conclude that our ADD-XOR and S-Box-CBC constructions have approximately $1/4-1/3$ of PHOTON's total circuit area (this also includes the control circuitry besides PHOTON) while maintaining an appropriate security level which takes care of economically motivated adversaries.

Keywords: Light-weight Cryptography; Unforgeability; One-Time Hash Function; Secure Supply Chain Management

1 Introduction

According to a recent report (in 2016) by the OECD and the EUs Intellectual Property Office [1], the value of imported counterfeited and pirated goods is worth nearly half a trillion dollars a year, which is around 2.5% of global imports with many of the proceeds going to organized crime. Close to 5% of goods that are imported into the European Union are fakes. The report analyses about half a million customs seizures around the world during 2011-13 covering all kinds of physical counterfeit goods (that infringe trademarks, intellectual property rights, or copyright) in order to obtain rigorous estimates of the scale of trade in counterfeit and pirated goods (online piracy is not included). These fake products

appear everywhere – the most dangerous ones are auto parts that fail, drugs making people sick, medical instruments delivering false readings, etc.

It is of utmost importance to make supply chains secure in order to detect counterfeit product injection into the supply chain. To this purpose, Radio-Frequency Identification (RFID) tags are used as a low-cost wireless identification method: Each product is equipped with an RFID tag which has a unique identifier and which is initialized at the supply chain back-end server with corresponding product information. In addition the RFID tag is either initialized with digital keys used for future authentication or, if a Physical Unclonable Function (PUF) is embedded, then read-out 'challenge response pairs' from the PUF at the back-end server can later be used for authentication. A supply chain moves through different supply chain partners that interact with RFID tags using RFID readers. These interactions are collected/stored and are together analyzed at the back-end server in order to detect whether the product is genuine or fake before the product exits the supply chain.

1.1 NVM-Based Scheme

This paper focuses on the most recent state-of-the-art proposal by Maleki et al. [2] for secure supply chain management based on RFID tags. Previous schemes come in two kinds: A first kind [3–7] requires persistent online communication between readers of supply chain partners and the back-end server. This, however, is in practice not always possible; sometimes the online connection does get lost and even an hour communication disruption can delay product transportation leading to financial loss. In order to avoid the need for persistent online communication, a second kind of scheme has been proposed which requires each supply chain partner to implement a local database [8–11]. Local databases are used as temporal storage to keep track of reader events. The local databases are integrated into the back-end server at a suitable time when an online connection with the back-end server is available.

The disadvantage of using local databases is that they need a reliable infrastructure and they must be maintained and secured. This imposes extra costs to partners and makes the supply chain possibly less secure as these local databases become an accessible point of attack. The main contribution of [2] is a new method which does not require any persistent online communication and also does not require local databases. Their idea is to distribute the local databases into the RFID tags themselves by utilizing the 2-3K bit Non-Volatile Memory (NVM) present in current off-the-shelve RFID tags [12]. This memory is sufficient for storing all the reader events the RFID tag engages in. Only when exiting the supply chain, the RFID tag is read out and verified by the back-end server for which online communication is needed (a minimal requirement for any scheme). The NVM-based scheme of Maleki et al. [2] is presented in detail in Appendix A (with discussion of pros and cons).

1.2 Software Unclonable Functions

The main take-away of the NVM-based scheme is that in order to bind the identity of an RFID tag to a reader event, the tag consumes one of its secret keys k stored in its NVM in order to compute $F_k(x)$ where input x is received from the reader and cannot be distinguished from a random bit string. The tag overwrites k with $F_k(x)$ in its NVM – and in this way the tag authenticates its own reader events stored in its NVM. For completeness, the reader represents the reader event (which includes the identities of the reader and tag, and a time stamp) as a bit string, which the reader MACs using its own key and this results in x. The back-end server has in its database the key sequence of the tag (which includes k) and the reader key; this is sufficient to verify the binding of the event to the reader (through the MAC) and tag (through $F_k(x)$).

Maleki et al. [2] explain that it is sufficient to require that $F_k(.)$ is a collision resistant hash function – and they propose to use PHOTON 80/20/16, a light-weight hash function costing 865 GE (Gate Equivalent). The complete solution (including control logic etc.) costs 1428 GE. Juels and Weis [13] state (and confirmed by [12]) "A basic RFID tag may have a total of anywhere from 1000-10000 gates, with only 200-2000 budgeted specifically for security." Also every 1000 gates costs approximately one dollar cent per tag. It is therefore important to further reduce the gate equivalence of the complete solution. It turns out that collision resistance is not required and this allows the design of a much more light-weight $F_k(.)$, which is the problem statement of this paper.

In the NVM-based scheme, $F_k(.)$ is used to protect against an adversary who can only access the tag through its read and write interface in order to gather sufficient information to construct a 'tag-simulator' which can be programmed into a fake tag. The read and write interface is such that after initialization only the values that have replaced keys in NVM can be read out. This means that in order to learn about one specific key k, an adversary can engage in a reader-like interaction with the tag in order to replace k with $F_k(x')$ for some value x' of his choice. Next $F_k(x')$ can be read out and the pair $(x', F_k(x'))$ can be used to design a simulator for predicting $F_k(x)$ for random input bit vectors x. Modeling this (very) weak attacker leads to the definition of "software unclonable functions" in [2] of which they only give collision resistant hash functions as an instance. We notice that in the discussion above the adversary is only allowed to use an RFID tag's read and write interface according to its specifications. An adversary who can circumvent the interface circuitry by means of a physical attack is not considered.

1.3 Contributions and Organization

In Section 2 we take the definition of software unclonable functions and give an equivalent definition in terms of a "software unclonable response game". Next we discuss its relation to the standard crypto notion of unforgeability for MACs and we conclude that being software unclonable means "unforgeable for random inputs when given one chosen input-output pair" – hence, the title of our paper.

We enrich the definition of software unclonability by adding a security measure for worst-case scenarios in concrete parameter settings.

Sections 3, 4, and 5 provide very light-weight constructions. The first, called MULTIPLY-ADD, is based on a simple multiplication with addition over integers. The second, called ADD-XOR, combines xor with addition-with-carry over binary vectors. The third, called S-Box-CBC, uses the idea of Cipher Block Chaining (CBC) mode with a specially designed S-box. In Section 6 we compare the different solutions and we show ADD-XOR and S-Box-CBC lead to dramatic reductions in total circuit size for reasonable security.

2 Software Unclonable Functions

Software unclonable functions are defined as follows:

Definition 1. *[2] A keyed function $F_k(.)$ is called software unclonable if the probability of guessing the output of $F_k(x)$ for a randomly chosen input x with knowledge of* **one** *chosen input-output pair $(x', F_k(x'))$ (the adversary chooses x') is less than negligible (in the function's key size).*

We notice that Definition 1 requires resistance against adversaries with unbounded computation – the definition formulates software unclonability in terms of information theoretic security. This implies that a symmetric key encryption scheme may not satisfy software unclonability since one chosen plaintext ciphertext pair may reveal significant information about the underlying secret key in the information theoretical setting. For a polynomial time adversary a semantically secure symmetric key encryption scheme will be software unclonable. Therefore, we recast the above definition for adversaries with polynomial computation by using the software unclonable response game given in Algorithm 1, where

- λ is a security parameter and $Gen(1^\lambda)$ is a ppt algorithm which generates a random key k,
- $\mathcal{A}_0$ is a ppt adversarial algorithm which allows the adversary to generate exactly one chosen input value x'
- for which the adversary is allowed to learn the output/response of function $F_k(x')$,
- $\mathcal{A}_1$ is a ppt adversarial algorithm which takes just this one input-output pair $(x', F_k(x'))$ in order to produce a ppt simulator $\mathcal{S}$, and
- where $\mathcal{S}$ successfully predicts $F_k(x)$ for a random input x if it outputs $F_k(x)$.

This leads to the following definition which we will use in our analysis in next sections:

Definition 2. *A function $F_k(x)$ is software unclonable if for any ppt pair $(\mathcal{A}_0, \mathcal{A}_1)$, the probability that* SoftwareUnclRespGame*$(\mathcal{A}_0, \mathcal{A}_1)$ (see Algorithm 1 with security parameter λ) returns 1 is $negl(\lambda)$.*

Algorithm 1 Software Unclonable Response Game

function SoftwareUnclRespGame($\mathcal{A}_0, \mathcal{A}_1$)
  $k \leftarrow \text{Gen}(1^\lambda)$;
  $x' \in \{0,1\}^\lambda \leftarrow \mathcal{A}_0(1^\lambda)$;
  $\mathcal{S} \leftarrow \mathcal{A}_1(x', F_k(x'))$; /* $\mathcal{S}$ is a ppt algorithm */
  Random $x \in \{0,1\}^\lambda$
  if $F_k(x) \leftarrow \mathcal{S}(x)$ **then** $b = 1$; **else** $b = 0$; **end if**
  Return b;
end function

Now, based on standard cryptography there are numerous cryptographic functions which are software unclonable. Given our motivation we are only interested in light-weight solutions in the sense that at most a couple 100 gates should suffice for implementation. This implies that the primitive (as far as the authors know) cannot be based on a computational hardness assumption. And this means that we need to prove information theoretical security for our constructions after all (in Definition 2 algorithms $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{S}$ do not need to be restricted to ppt). As a consequence, since the adversary can use one λ-bit vector equation representing an input-output pair for construction of a simulator, the key must have size at least $\lambda + O(\lambda)$ (as opposed to a symmetric key encryption scheme which can have a λ-bit secret key).

2.1 Unforgeability

A closer look at Definition 2 shows the relation of a software unclonable function to a Message Authentication Code (MAC): A MAC is a triple $(Gen, Sign, Ver)$ of ppt algorithms where $k \leftarrow Gen(1^\lambda)$ with security parameter λ, $t \leftarrow Sign(k, x)$ produces a tag t for input string x with key k, and Ver verifies whether a tag t fits input x with key k. A software unclonable function $F_k(x)$ plays the role of producing tags as in $Sign$. A first small difference is that $F_k(.)$ is a function and not an algorithm. This implies that verification of the tag is straightforward in that the corresponding Ver simply verifies whether tag $t = F_k(x)$ (and this directly implies the correctness property for a MAC).

The security of a MAC is defined as follows: A MAC is unforgeable if for all ppt algorithms $\mathcal{A}$,

$$Prob\left(\begin{array}{l} k \leftarrow Gen(1^\lambda), \\ (x,t) \leftarrow \mathcal{A}^{Sign(k,.)}(1^\lambda) \text{ where } \mathcal{A} \text{ does not query } Sign(k,x), \\ Ver(k,x,t) = \text{accept} \end{array}\right) < negl(\lambda),$$

where $\mathcal{A}^{Sign(k,.)}$ denotes that $\mathcal{A}$ has access to oracle $Sign(k,.)$.

The second difference with software unclonability is that the adversarial algorithm is split into $\mathcal{A}_0$, which selects an x' for querying oracle $Sign(k,.) = F_k(.)$, and $\mathcal{A}_1$ which, based on the output of the queried oracle, produces a simulator $\mathcal{S}$ whose goal is to produce a valid (verifiable) tag $Sign(k,x) = F_k(x)$ for

some random input x. This means that software unclonability does not allow the adversary to adaptively choose an x for which a tag is constructed, instead unforgeability is for tags of random inputs.

The third difference is that software unclonability does not allow the adversary to have a polynomial number of queries to the oracle, instead only one chosen input-tag pair can be used. We conclude that a software unclonable function produces tags as in a MAC with a much weaker unforgeability property: a tag corresponding to a random input is unforgeable given only one chosen input-tag pair. The motivation presented in the introduction has led to the name "software unclonable". From a crypto perspective, however, a better terminology would be "unforgeable for random inputs when given one chosen input-output pair" and call the primitive a "one-time MAC for authenticating random (not chosen) inputs."

In a one-time MAC [14] a key is used at most once and can be constructed using a universal hash function which is pairwise independent. An example lightweight pairwise independent hash function is defined by tag $t = k_0 x + k_1 \bmod p$, where p is prime. In Section 3 we analyse the "MULTIPLY-ADD" function where a tag is computed as $k_0 x + k_1 \bmod 2^\lambda$ and the "key" (k_0, k_1) is chosen at random (not necessarily odd).

2.2 Average vs. Worst-Case Analysis

Software unclonable functions are meant to be applied in RFID-based secure supply chain management. Rather than just proving asymptotic results in the form of the probability of a successful attack being negligible in λ, we want to know a concrete upper bound on this probability as a function of λ. This will allow us to suggest concrete parameter settings.

As we will explain below, Definition 2 talks about the average over 'queries $x' \leftarrow \mathcal{A}_0(1^\lambda)$ to oracle $F_k(.)$' of the probability of a successful prediction by the simulator computed by $\mathcal{S} \leftarrow \mathcal{A}_1(x', F_k(x'))$. In asymptotic terms, if this average is negligible, then it is not possible to have a significant worst-case probability γ_0 of selecting an oracle query which leads to a simulator which also has a significant probability $\geq \gamma_1$ of success, because $\gamma_0\gamma_1$ is at most average p which is negligible,

$$\gamma_0\gamma_1 \leq p. \tag{1}$$

In other words, either γ_0 or γ_1 must be negligible.

In this argument we did not specify the pair (γ_0, γ_1) and we note that there are many possibilities. In the concrete non-asymptotic setting, we want an achievable pair (γ_0, γ_1) for which both the probability γ_0 of having a 'lucky' query as well as the probability γ_1 of succesfull prediction by a simulator originating from 'normal' queries to be equally small: If we find such a pair, then we know that both the worst-case probability γ_0 is small as well as the probability γ_1 of success in the normal case is small. This is not captured by studying the concrete asymptotic behavior of the average as a function of λ.

For example, if $p = 2^{-\lambda}$, then the minimum α of $\max\{\gamma_0, \gamma_1\}$ over all possible/achievable pairs (γ_0, γ_1) could be realized by $\gamma_0 = \gamma_1 = 2^{-\lambda/2}$ which meets

(1) with equality (the argument is in essence the application of the birthday paradox to our problem setting). This leads to a very different concrete parameter setting compared to 'just' considering the average case.

So, we are still not entirely satisfied with Definition 2 when considering concrete parameter settings for the following reason: For $\mathcal{A}_1$ and $x' \in \{0,1\}^\lambda$, let

$$p[\mathcal{A}_1](x') = Prob_{x \leftarrow \{0,1\}^\lambda}(F_k(x) \leftarrow \mathcal{S}(x) | \mathcal{S} \leftarrow \mathcal{A}_1(x')). \tag{2}$$

In Definiton 2 the probability $p[\mathcal{A}_0, \mathcal{A}_1]$ that SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ returns 1 (over random x and coin flips in $\mathcal{A}_0$, $\mathcal{A}_1$, and $\mathcal{S}$) is equal to the average

$$p[\mathcal{A}_0, \mathcal{A}_1] = \sum_{x' \in \{0,1\}^\lambda} Prob(x' \leftarrow \mathcal{A}_0(1^\lambda)) p[\mathcal{A}_1](x'). \tag{3}$$

In Definition 2 we only require that the average $p[\mathcal{A}_0, \mathcal{A}_1]$ should be negligible in λ. As explained above, we need to formulate security in terms of a worst-case analysis. We may ask whether the adversary can be lucky (the worst-case) and somehow select in $\mathcal{A}_0$ a x' which "fits" k well in that $p[\mathcal{A}_1](x')$ is (much) larger than the average $p[\mathcal{A}_0, \mathcal{A}_1]$. In order to analyze this we introduce $\alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ as the probability (over coin flips used in $\mathcal{A}_0$ and $\mathcal{A}_1$) that game SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ produces a simulator $\mathcal{S}$ which correctly predicts $F_k(x) \leftarrow \mathcal{S}(x)$ with probability (over random x and coin flips in $\mathcal{S}$) $\leq 2^{-h}$. We note that

$$\alpha_h[\mathcal{A}_0, \mathcal{A}_1] = \sum_{x' : p[\mathcal{A}_1](x') \leq 2^{-h}} Prob(x' \leftarrow \mathcal{A}_0(1^\lambda)). \tag{4}$$

We want both 2^{-h} small and $\alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ large as this implies that (1) the probability that the adversary is able to construct a "lucky" simulator is equal to $1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1]$, which is small, and (2) when the adversary constructs a "normal" (i.e., "not lucky") simulator, then the simulator correctly predicts $F_k(x) \leftarrow \mathcal{S}(x)$ with small probability $\leq 2^{-h}$. We can think of $1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ as the probability mass of the tail of distribution $p[\mathcal{A}_1](x')$ that describes the lucky scenarios x' for the adversary, i.e., the worst-case scenarios from a security point of view.

We want $1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1]$ and 2^{-h} to be in balance and this leads to the definition of

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] = \min_h \max\{1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1], 2^{-h}\}. \tag{5}$$

$\alpha[\mathcal{A}_0, \mathcal{A}_1]$ is the smallest value α with the property that the probability that game SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ produces a simulator $\mathcal{S}$ which correctly predicts $F_k(x) \leftarrow \mathcal{S}(x)$ with probability $> \alpha$ is at most α, in formula,

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] = \min \left\{ \alpha : Prob \left(\begin{array}{l} \mathcal{S} \leftarrow \mathcal{A}_1(x') \text{ such that} \\ Prob(F_k(x) \leftarrow \mathcal{S}(x)) > \alpha \end{array} \right) \leq \alpha \right\},$$

where the inner probability is over random $x \leftarrow \{0,1\}^\lambda$ and the outer probability is over $x' \leftarrow \mathcal{A}_0(1^\lambda)$.

Definition 3. *For a software unclonable function $F_k(x)$ we define the 'average exponential growth factor'*

$$\mathbf{p} = \limsup_{\lambda\to\infty} -(\log \sup_{(\mathcal{A}_0,\mathcal{A}_1)} p[\mathcal{A}_0,\mathcal{A}_1])/\lambda$$

and we define the 'worst-case exponential growth factor'

$$\mathbf{a} = \limsup_{\lambda\to\infty} -(\log \sup_{(\mathcal{A}_0,\mathcal{A}_1)} \alpha[\mathcal{A}_0,\mathcal{A}_1])/\lambda,$$

where $p[\mathcal{A}_0,\mathcal{A}_1]$ is a function of λ given by (2,3) and $\alpha[\mathcal{A}_0,\mathcal{A}_1]$ is a function of λ given by (2,4,5).

A software unclonable function $F_k(x)$ has better security if $\mathbf{a}$ is larger, and is more light-weight if the gate equivalence of its circuit implementation is smaller. In this paper we propose three constructions and compare them along these metrics.

Notice that by (3, 4), $p[\mathcal{A}_0,\mathcal{A}_1] \geq (1-\alpha_h[\mathcal{A}_0,\mathcal{A}_1])2^{-h}$. Combined with (5), this implies

$$\alpha[\mathcal{A}_0,\mathcal{A}_1]) \leq \min_h \max\{p[\mathcal{A}_0,\mathcal{A}_1]/2^{-h}, 2^{-h}\} = \sqrt{p[\mathcal{A}_0,\mathcal{A}_1]}$$

(since h can be any real number; in the analysis of our constructions we consider only integers h). This proves that the exponential growth factors $\mathbf{p}$ and $\mathbf{a}$ satisfy $\mathbf{a} \geq \mathbf{p}/2$. This argument is in essence the birthday paradox.

In the next sections we analyze for several candidate software unclonable functions both $p[\mathcal{A}_0,\mathcal{A}_1]$ as well as $\alpha[\mathcal{A}_0,\mathcal{A}_1]$ together with their exponential growth factors. It turns out that for these functions the probability mass of the tail of distribution $p[\mathcal{A}_0](x')$ is large so that $\mathbf{a} \approx \mathbf{p}/2$.

3 MULTIPLY-ADD

Below we prove that the "MULTIPLY-ADD" function

$$F_{(k_0,k_1)}(x) = k_0 x + k_1 \bmod 2^\lambda, \text{ for } k_0,k_1,x \in \{0,1\}^\lambda, \tag{6}$$

where k_0 and x are multiplied modulo 2^λ and $+$ modulo 2^λ is binary addition with carry truncated after λ bits, is a software unclonable function. In what follows when we write $+$ we mean addition modulo 2^λ.

Theorem 1. *For the MULTIPLY-ADD function defined in (6),*

$$p[\mathcal{A}_0,\mathcal{A}_1] \leq (\lambda+2)2^{-\lambda-1} \text{ and } \alpha[\mathcal{A}_0,\mathcal{A}_1] \leq 2^{-\lfloor(\lambda-1)/2\rfloor}$$

for all algorithm pairs $(\mathcal{A}_0,\mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). These upper bounds can be met with equality, this implies average and worst-case exponential growth factors

$$\mathbf{p} = 1 \text{ and } \mathbf{a} = 1/2.$$

Proof. We will first translate the problem of creating a simulator with maximum possible successful prediction probability to an equivalent problem which we are able to analyze precisely:

Suppose the adversary knows the pair $(x', F_{(k_0,k_1)}(x'))$ and wants to built a simulator which predicts $F_{(k_0,k_1)}(x)$ for random x. We notice that for

$$z = F_{(k_0,k_1)}(x) - F_{(k_0,k_1)}(x'), \quad v = k_0, \text{ and } w = x - x',$$

$$z = vw \bmod 2^\lambda. \tag{7}$$

Also, notice that given x', since x is random, w is random; and since k_0 is random, v is random. These observations can be used to show that predicting $F_{(k_0,k_1)}(x)$ for a randomly selected input x based on $(x', F_{(k_0,k_1)}(x'))$ where F is defined by (6) is equivalent to (notice that k_1 is unknown and random) predicting z in (7) for a randomly selected input w and unknown/random v. This implies that the probability of SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ returning 1 is equal to the probability of ZWGameMA$(\mathcal{A})$, see Algorithm 2, returning 1. This probability is maximized for $\mathcal{A}$ outputting a simulator $\mathcal{S}$ which on input w outputs a z that maximizes $|\{v : z = vw\}|$. In other words, z maximizes the number of collisions v that yield the same $z = vw$. In formula, $p[\mathcal{A}_0, \mathcal{A}_1]$ (where $\mathcal{A}_0$ and $\mathcal{A}_1$ are derived from $\mathcal{A}$ according to the transformation described above) is equal to

$$2^{-\lambda} \sum_w \max_z Prob_{v \leftarrow \{0,1\}^\lambda}[z = vw] = 2^{-\lambda} \sum_w \max_z \frac{|\{v : z = vw\}|}{2^\lambda}. \tag{8}$$

Algorithm 2 Finding z based on w

```
1: function ZWGameMA(A)
2:     v ∈ {0,1}^λ is a random input;
3:     S ← A(1^λ);
4:     w ∈ {0,1}^λ is a random input;
5:     if S(w) = vw then
6:         b = 1;
7:     else
8:         b = 0;
9:     end if
10:    Return b;
11: end function
12: /*On input w, an optimal S outputs a z which maximizes |{v : z = vw}|.*/
```

We will now analyze (8) by distinguishing the cases $w \neq 0$ and $w = 0$.

Let $w \neq 0$. If $2^{\lambda-h}$ is the largest power of 2 dividing w, then $z = vw \bmod 2^\lambda$ is equivalent to $z = v(w/2^{\lambda-h}) \bmod 2^h$. Since $w/2^{\lambda-h}$ is an odd integer, it has an inverse modulo 2^h. This implies that there is exactly one $v = z(w/2^{\lambda-h})^{-1} \bmod 2^h$ for which $z = v(w/2^{\lambda-h}) \bmod 2^h$. Therefore there are $2^{\lambda-h}$ possible v for

which $z = vw \bmod 2^\lambda$ (these v are equal to $z(w/2^{\lambda-h})^{-1} \bmod 2^h$ plus some multiple of 2^h).

If $w = 0$, then only for $z = 0$ there exists a v such that $z = vw$; in this case all 2^λ possible v satisfy $z = vw$.

Let W_h, $1 < h \leq \lambda$, be the number of integers w, $0 < w < 2^\lambda$, for which $2^{\lambda-h}$ is the largest power of 2 dividing w. Define $W_0 = 1$. Then (8) is equal to

$$2^{-\lambda} \sum_{h=0}^{\lambda} W_h 2^{-h}.$$

We notice that $W_h = 2^{h-1}$ for $1 \leq h \leq \lambda$. Hence, (8) is equal to

$$2^{-\lambda}(1 + \sum_{h=1}^{\lambda} 2^{-1}) = (\lambda + 2)2^{-\lambda-1}.$$

This proves $p[\mathcal{A}_0, \mathcal{A}_1] = (\lambda + 2)2^{-\lambda-1}$ and $\mathbf{p} = 1$.

The above derivation also proves that the number of w for which $\max_z Prob_{v \leftarrow \{0,1\}^\lambda}[z = vw] \leq 2^{-h}$ is equal to $\sum_{i=h}^{\lambda} W_i = 2^\lambda - \sum_{i=0}^{h-1} W_h = 2^\lambda - (1 + \sum_{i=1}^{h-1} 2^{i-1})$ for $h \geq 2$. The probability that such a w is selected is equal to $\sum_{i=h}^{\lambda} W_i/2^\lambda$. This can be interpreted as

$$\alpha_h[\mathcal{A}_0, \mathcal{A}_1] = \sum_{i=h}^{\lambda} W_i/2^\lambda = 1 - (1 + \sum_{i=1}^{h-1} 2^{i-1})/2^\lambda = 1 - 2^{-(\lambda+1-h)},$$

which in turn proves

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] = \min_h \max\{2^{-(\lambda+1-h)}, 2^{-h}\} = 2^{-\lfloor(\lambda-1)/2\rfloor} \text{ and } \mathbf{a} = 1/2.$$

4 ADD-XOR

Below we prove that the "ADD-XOR" function

$$F_{(k_0,k_1)}(x) = (k_0 + x \bmod 2^\lambda) \oplus k_1, \text{ for } k_0, k_1, x \in \{0,1\}^\lambda, \tag{9}$$

where $+$ modulo 2^λ is binary addition with carry truncated after λ bits and where $\oplus$ represents XOR, is a software unclonable function.

In appendix B, we prove the following theorem:

Theorem 2. *For the ADD-XOR function as defined in (9),*

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq 2^{-0.234 \cdot \lambda} \text{ and } \alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2 \cdot 2^{-0.141 \cdot \lambda}$$

for all algorithm pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). This implies average and worst-case exponential growth factors

$$\mathbf{p} \geq 0.234 \text{ and } \mathbf{a} \geq 0.141.$$

Simulations with the optimal simulator constructed in Appendix B show that the bounds for $\mathbf{p}$ and $\mathbf{a}$ are very tight. (Notice that for ADD-XOR, $\mathbf{a} > \mathbf{p}/2$.)

5 S-Box-CBC

In this section we introduce a construction which uses the idea of S-boxes in block-cipher design together with CBC mode [15].

Suppose we have a non-linear mapping

$$S \in \{0,1\}^m \to \{0,1\}^m,$$

where m is generally very small (we will propose small powers of two, $m = 4$ or $m = 8$). Mapping S is called an S-Box. Since we use a software unclonable function for authentication in the NVM-based supply chain management scheme, this means that the software unclonable function does not necessarily need to be invertible given knowledge of the keys. It turns out that ADD-XOR and MULTIPLY-ADD are invertible; in this section, however, we will construct a non-invertible software unclonable function based on a non-invertible S-box mapping S.

Our construction is iterative following the design principle used in CBC mode for symmetric key encryption (where we replace encryption by our S-box): For n with $nm = \lambda$, we use the vector notation $x = (x_1, \ldots, x_n) \in \{0,1\}^\lambda$ with $x_i \in \{0,1\}^m$. For keys $k_0 = (k_1^0, k_2^0, \ldots, k_n^0)$ and $k_1 = (k_1^1, k_2^1, \ldots, k_n^1)$ and input x, we recursively compute

$$y_{i+1} = S(y_i \oplus x_{i+1} \oplus k_{i+1}^0) \oplus k_{i+1}^1 \tag{10}$$

for $0 \leq i \leq n-1$ with $y_0 = 0$. We define

$$F_{(k_0,k_1)}(x) = y. \tag{11}$$

In the construction we mask input x_i with k_i^0 and we mask the output of the S-box with k_i^1. The S-box is a kind of non-linear obfuscation mapping. Forwarding y_i into the computation of y_{i+1} corresponds to the main design principle used in CBC mode for symmetric key encryption. Below we will prove (in a couple of steps) that the S-box construction leads to a software unclonable function.

We start with analyzing the average case:

Theorem 3. *Let $F_{(k_0,k_1)}(x)$ be defined by the S-Box-CBC construction in (10,11) for $\lambda = nm$. For the S-box mapping S used in F, we define*

$$\rho[S](w) = \max_{z \in \{0,1\}^m} |\{v : z = S(v) \oplus S(v \oplus w)\}|/2^m, \text{ and}$$
$$\rho[S] = \sum_{w \in \{0,1\}^m} \rho[S](w)/2^m.$$

Then, for all algorithm pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt),

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq \rho[S]^n \text{ with } \mathbf{p} = -(\log \rho[S])/m.$$

Proof. We will first translate the problem of creating a simulator with maximum possible successful prediction probability to an equivalent problem which we are able to analyze precisely:

Suppose the adversary knows the pair $(x', y' = F_{(k_0,k_1)}(x'))$ and wants to built a simulator which predicts $F_{(k_0,k_1)}(x)$ for random x. We notice that for $z = F_{(k_0,k_1)}(x') \oplus F_{(k_0,k_1)}(x)$ the recursive definition of F in (10,11) implies $z_{i+1} = y'_{i+1} \oplus y_{i+1} = S(y'_i \oplus x'_{i+1} \oplus k^0_{i+1}) \oplus S(y_i \oplus x_{i+1} \oplus k^0_{i+1})$. If we define $v_{i+1} = y'_i \oplus x'_{i+1} \oplus k^0_{i+1}$, and $w_{i+1} = (y'_i \oplus y_i) \oplus (x'_{i+1} \oplus x_{i+1}) = z_i \oplus (x'_{i+1} \oplus x_{i+1})$, then

$$z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1}). \tag{12}$$

Notice that given x'_{i+1} and y'_i, since k^0_{i+1} is random and y'_i only depends on k^0_j for $j \leq i$, v_{i+1} is random. Therefore, by induction on i, given x', v is random. We also notice that given x'_{i+1} and z_i, since x_{i+1} is random and z_i only depends on x_j for $j \leq i$, w_{i+1} is random. Therefore, by induction on i, given x', w is random.

The above observations can be used to show that predicting $F_{(k_0,k_1)}(x)$ for a randomly selected input x based on $(x', F_{(k_0,k_1)}(x'))$ where F is defined by (10,11) is equivalent to (notice that k_1 is unknown and random) predicting z in (12) for a randomly selected input w and unknown/random v. This implies that the probability of SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ returning 1 is equal to the probability of ZWGameSB$(\mathcal{A})$, see Algorithm 3, returning 1. This probability is maximized over $\mathcal{A}$ outputting a simulator $\mathcal{S}$ which on input w outputs a z that maximizes $|\{v : \forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})\}|$. In other words, z maximizes the number of collisions v that satisfy the same set of equations $z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})$. In formula, $p[\mathcal{A}_0, \mathcal{A}_1]$ (where $\mathcal{A}_0$ and $\mathcal{A}_1$ are derived from $\mathcal{A}$ according to the transformation described above) is equal to

$$\begin{aligned}
&2^{-\lambda} \sum_w \max_z Prob_{v \leftarrow \{0,1\}^\lambda}[\forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})] \\
&= 2^{-\lambda} \sum_w \max_z \frac{|\{v : \forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})\}|}{2^\lambda}. \\
&= 2^{-\lambda} \sum_w \prod_{i=0}^{n-1} \rho[S](w_{i+1}) = \prod_{i=0}^{n-1} \sum_{w_{i+1}} \rho[S](w_{i+1})/2^m = \rho[S]^n.
\end{aligned} \tag{13}$$

This concludes the proof of Theorem 3.

According to Theorem 3, the smaller $\rho[S]$, the larger the average exponential growth factor $\mathbf{p}$. The next lemma shows a lower bound on $\rho[S]$ and describes an S-box $\mathcal{S}$ which meets this lower bound leading to the largest possible $\mathbf{p}$ for the S-Box-CBC construction:

Lemma 1. *(i) For any S-box S, $\rho[S](w) \geq 2/2^m$ for $w \neq 0$. If $w = 0$, then $\rho[S](w) = 1$. As a consequence $\rho[S] \geq (3 - 1/2^{m-1})2^{-m}$. This lower bound can be met with equality: (ii) Let $m \geq 3$. If we represent elements in $\{0,1\}^m$ as finite field elements in $GF(2^m)$ and define $S(x) = x^3$ in $GF(2^m)$, then $\rho[S](w) = 2/2^m$ for $w \neq 0$ and $\rho[S] = (3 - 1/2^{m-1})2^{-m}$.*

Algorithm 3 Finding z based on w

function ZWGAMESB($\mathcal{A}$)
  $v \in \{0,1\}^\lambda$ is a random input;
  $\mathcal{S} \leftarrow \mathcal{A}(1^\lambda)$;
  $w \in \{0,1\}^\lambda$ is a random input;
  if $\forall_i z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})$ **then**
    $b = 1$;
  else
    $b = 0$;
  end if
  Return b;
end function
/*On input w, an optimal $\mathcal{S}$ outputs a z which maximizes $|\{v : \forall_i\ z_{i+1} = S(v_{i+1}) \oplus S(v_{i+1} \oplus w_{i+1})\}|$.*/

Proof. Let $z, v, w \in \{0,1\}^m$ (in the proof of the previous theorem $z, v, w \in \{0,1\}^\lambda$). The first part of the lemma follows immediately from the observation that if $z = S(v) \oplus S(v \oplus w)$ then $v' = v \oplus w$ also satisfies this equation. If $w \neq 0$, then $v \neq v'$ and this shows that if a solution v for $z = S(v) \oplus S(v \oplus w)$ exists, then there also exists a second different solution, hence, $\rho[S](w) \geq 2/2^m$. If $w = 0$, then $S(v) \oplus S(v \oplus w) = S(v) \oplus S(v) = 0$ for all v. As a consequence $\rho[S](0) = 2^m/2^m = 1$ and $\rho[S] \geq 1/2^m + (1 - 1/2^m)2/2^m = (3 - 1/2^{m-1})2^{-m}$.

In the second part of the lemma we take $S(x) = x^3$ where we consider binary vectors $x \in \{0,1\}^m$ to represent elements in $GF(2^m)$. Notice that $\oplus$ becomes addition in $GF(2^m)$. This means that the equation $z = S(v) \oplus S(v \oplus w)$ translates to $z = v^3 + (v + w)^3 = v^2 w + v w^2 + w^3$ in $GF(2^m)$. This is equivalent to

$$w v^2 + w^2 v + (w^3 + z) = 0,$$

a quadratic equation in v for $w \neq 0$ (notice that w^2 does not reduce to a linear expression in w since the irreducible polynomial defining $GF(2^m)$ has degree ≥ 3 for $m \geq 3$). If $w \neq 0$, then the equation becomes $v^2 + wv + (w^2 + z w^{-1}) = 0$ which has at most 2 solutions, hence, $\rho[S](w) = 2/2^m$.

Corollary 1. *For the S-Box-CBC construction in (10,11) for $\lambda = nm$ and the S-box specified in Lemma 1(ii),*

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq (3 - 1/2^{m-1})^{\lambda/m} 2^{-\lambda} \text{ and } \mathbf{p} = 1 - \frac{\log(3 - 1/2^{m-1})}{m}$$

for all algorithms pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). There exist algorithm pairs for which the upper bound holds with equality.

We prove our result for the worst-case scenario in Appendix C.

Theorem 4. *For the S-Box-CBC construction in (10,11) for $\lambda = nm$ and S-box $\mathcal{S}$ specified in Lemma 1(ii),*

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \frac{\lambda(m-1)}{m(2m-1)} 2^{-\frac{(m-1)^2}{m(2m-1)}\lambda} \text{ with } \mathbf{a} = \frac{(m-1)^2}{m(2m-1)},$$

for all algorithms pairs $(\mathcal{A}_0, \mathcal{A}_1)$ (unrestricted, the theorem does not require $(\mathcal{A}_0, \mathcal{A}_1)$, and produced simulators to be ppt). There exist algorithm pairs for which the upper bound is almost tight. Notice that **a** *is only slightly larger than* $\mathbf{p}/2$.

6 Comparison

Our theorems state for unbounded adversaries (we prove information theoretical security as opposed to PHOTON which assumes computational hardness)

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2^{-\lfloor(\lambda-1)/2\rfloor} \text{ for MULTIPLY-ADD,}$$
$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2 \cdot 2^{-0.141 \cdot \lambda} \text{ for ADD-XOR, and}$$
$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \frac{\lambda(m-1)}{m(2m-1)} 2^{-\frac{(m-1)^2}{m(2m-1)}\lambda} \text{ for S-Box-CBC.}$$

Table 1 lists for which λ the different constructions give rise to upper bounds approximately equal to 2^{-16}, 2^{-32}, 2^{-40}, and 2^{-64}.

$\alpha[\mathcal{A}_0, \mathcal{A}_1]$	MULT.-ADD	ADD-XOR	S-Box-CBC $m=4$	S-Box-CBC $m=8$	PHOTON
$\leq 2^{-16}$	$\lambda = 33$ 1200b NVM $\geq$ 726 GE	$\lambda = 121$ 2960b NVM 372 GE (308+64)	$\lambda = 60$ 1680b NVM 440 GE (277+163)	$\lambda = 48$ 1360b NVM 726 GE (261+465)	N/A
$\leq 2^{-32}$	$\lambda = 65$ 1840b NVM $\geq$ 1430 GE	$\lambda = 243$ **5360**b NVM	$\lambda = 112$ 2640b NVM 478 GE (315+163)	$\lambda = 88$ 2160b NVM 748 GE (283+465)	N/A
$\leq 2^{-40}$	$\lambda = 81$ 2160b NVM $\geq$ 1782 GE	$\lambda = 291$ **6320**b NVM	$\lambda = 140$ **3280**b NVM	$\lambda = 112$ 2640b NVM 764 GE (299+465)	$\lambda = 80$ 1200b NVM 1428 GE [2] (563+865)
$\leq 2^{-64}$	$\lambda = 129$ **3120**b NVM	$\lambda = 461$ **9680**b NVM	$\lambda = 216$ **4720**b NVM	$\lambda = 168$ **3760**b NVM	$\lambda = 128$ 1680b NVM 1795 GE (673+1122)

Table 1. Comparison light-weight constructions.

Since we want to prevent an adversary from successfully cloning an RFID tag in order to tag and insert a counterfeit product into the supply chain, we

are actually fine with very small unconditional collision resistance. Even 2^{-16} is acceptable for the following reason: Our constructions are unconditional secure in that even an adversary with unbounded computational resources cannot create a cloned RFID tag which can do better than answering future reader queries with probability $> 2^{-16}$. This implies that only one out of 64K inserted counterfeit products makes it successfully through the supply chain. This is an economical impractical proposition for the adversary.

One reason to have a higher collision resistance like 2^{-32} is if fake trapdoored products are very cheap and the adversary wants to disrupt consumers of these products, e.g., the military. In this case the adversary is not economically motivated, he simply wants to be able to get a hardware footprint in order to be able to launch future attacks. In this case a collision resistance of 2^{-32} would imply an enormous number (4 billion) fake products needed by the adversary making such an attack quite impractical. When we put our crypto-hat on, we may even want the psychological safe collision resistance of 2^{-64}.

For each solution, it is important to verify whether it fits the 2-3K bit NVM requirement. A reader event is 40 bits [2] with a λ-bit output of the software unclonable function, which will replace one 40-bit key for a one time pad in the NVM based scheme and two λ-bit keys for our software unclonable constructions – while PHOTON only needs one λ-bit key. Since the RFID NVM is assumed to be read out per byte, we will round λ bits up to a multiple of bytes. Hence, for each reader event, a total of $5 + 2\lceil\lambda/8\rceil\}$ bytes in NVM are needed. Following [2], we expect at most 10 reader events per path through the supply chain. This gives a total of $10 \cdot 8 \cdot (5 + 2\lceil\lambda/8\rceil)$ required NVM bits. In bold are indicated which entries violate the 2-3K bit requirement.

Table 1 also compares constructions (that do not violate the 2-3K bit NVM requirement) with respect to the Gate Equivalence (GE) – measured in number of 2-input NAND gates – of their circuit implementations[1]. In the table "261+465" under e.g. the S-Box-CBC entry for $m = 8$ and $\lambda = 48$ indicates that 465 GE is spend on the software unclonable function with its own control logic and another 261 GE is spend on general control logic for a full NVM-based supply chain management scheme implementation; $261 + 465$ GE makes a total of 726 GE. Appendix D lists and explains the optimal implementations of each construction – we list the implemented results for ADD-XOR and S-Box-CBC, and the estimated lower bound $\geq 22 \cdot \lambda$ GE for the MULTIPLY-ADD construction.

PHOTON [16] has two light-weight variations: [2] uses PHOTON 80/20/16 as software unclonable function. It takes 80 bits input and generates 80 bits output with 40 bits collision resistance considering state-of-the-art attacks[2]. Table 1 also shows PHOTON 128/16/16 with 64 bits collision resistance.

The shaded entries in Table 1 minimize the circuit area given a 2-3K bit NVM constraint.

[1] GE is a metric for comparing the size of hardware implementation regardless of the manufacturing technology.

[2] [2] states 64 bits collision resistance, but this is incorrect.

7 Conclusion

We introduced a formal definition of software unclonable functions and constructed several light-weight options with a rigorous security analysis. Our ADD-XOR and S-Box-CBC ($m = 4$) constructions significantly reduce the circuit size of the implementation of the NVM-based supply chain management scheme of Maleki et al. [2] to 372 GE for $\alpha \leq 2^{-16}$ and 478 GE for $\alpha \leq 2^{-32}$. When compared to PHOTON, S-Box-CBC ($m = 8$) gives the smaller area size of 764 GE for $\alpha \leq 2^{-40}$, while PHOTON 128/16/16 with 1795 GE is the preferred choice for very small $\alpha \leq 2^{-64}$. For an economically motivated adversary, it turns out that $\alpha \leq 2^{-16}$ offers sufficient protection.

A NVM-Based RFID Scheme

The NVM-based scheme of Maleki et al. [2] implements the following steps:

Initialization RFID tag The NVM of the RFID tag is initialized with a sequence of keys $(k^1, k^2, \ldots, k^u)$ and a pointer $p = 1$. The back-end server stores the RFID identity ID together with the sequence of keys.

Initialization Reader Each RFID tag reader is initialized with its own key K. The back-end server stores the reader identity together with K.

Reader Event The RFID tag is read out by a reader of a supply chain partner:

1. The RFID tage transmits its ID to the reader.
2. The reader creates a message m which has the reader identity and time stamp of the event.
3. The reader computes $x = MAC_K(m, ID)$. This binds the reader to the event.
4. The reader transmits (m, x) to the RFID tag.
5. The RFID tag receives (m, x). This triggers the RFID tag to read a next key k^p from its NVM and to increment pointer p by 1. Key k^p is large enough in order to be split up into a first part $k^{p,0}$ and a second part $k^{p,1}$. The tag computes the pair $y^p = (m \oplus k^{p,0}, F_{k^{p,1}}(x))$, where F is a software unclonable function. Since k^p is unique to the RFID tag, the use of function F binds the RFID tag to the event. The key part $k^{p,0}$ serves as a one time pad which prevents traceability.
6. The RFID tag stores y^p at the spot where k^p was stored in NVM.

Exit When the tag exits the supply chain, its NVM is read out and communicated to the back-end server. I.e., the internal logic only allows NVM to be read out up to but not including the address pointed at by pointer p. This means that the back-end server receives ID together with $(y^1, \ldots, y^{p-1})$. The ID is used to look up the sequence of keys corresponding to the tag. For each y, this allows the server to first reconstruct the messages m, second to extract the corresponding reader identity with key K from its database, third to compute the mac value $x = MAC_K(m, ID)$, fourth to evaluate F on x with the appropriate key, and finally verify that this is part of y. If all checks pass, then the recorded reader events were not impersonated and

they can be verified to correspond to a legitimate path through the supply chain. The server will invalidate the tag for future use in its database.

A detailed explanation, security analysis, and discussion around how to make the scheme reliable with respect to miss reads and miss writes can be found in [2].

A comparison of state-of-the-art schemes over a range of metrics can be found in [17]. Besides being unique in that, unlike any other scheme, the need for persistent online communication or local databases is avoided, the NVM-based scheme also compares well with most competitive other schemes. The only dimension on which the NVM-based scheme scores negatively is its lack of being able to resist physical attack (where a strong adversary attempts to circumvent the read and write interface in order to clone all the keys stored in NVM). We notice that even though the trace based scheme [8] can withstand physical attacks, the scheme cannot distinguish between a fake and legitimate tag which possibly results in significant financial loss. Current PUF based schemes [18–20] are not secure against physical attack because of recent machine learning modeling attacks [21–23] – however, as soon as improved PUF designs will resist these modeling attacks, PUF based schemes will resist physical attacks as opposed to the NVM-based scheme. Inherent to current PUF-based schemes, they do need persistent online communication. Also an improved PUF design will likely lead to a higher gate count than the 500-1000 GE for current PUF-based schemes – and this is where the NVM based scheme performs better as well.

As a final note, Section 6 discusses several upper bounds on the collision resistance. Obviously, if the resistance is set to 2^{-32} or 2^{-64}, then creating a cloned or fake RFID tag which successfully passes the supply chain becomes very unlikely. In fact too many counterfeit products labelled with fake RFID tags are needed in order to be successful and this makes such an attack economically infeasible. In the introduction we state "An adversary who can circumvent the interface circuitry by means of a physical attack is not considered." Clearly, the weak link in the NVM-based scheme for high collision resistance will now be its lack of resistance against physical attack.

B Security Analysis ADD-XOR

The proof of Theorem 2 consists of several steps. We will first translate the problem of creating a simulator with maximum possible successful prediction probability to an equivalent problem which we are able to analyze precisely:

Suppose the adversary knows the pair $(x', F_{(k_0,k_1)}(x'))$ and wants to build a simulator which predicts $F_{(k_0,k_1)}(x)$ for random x. We notice that for

$$z = F_{(k_0,k_1)}(x') \oplus F_{(k_0,k_1)}(x), \quad v = k_0 + x', \text{ and } v = x - x',$$

$$z \oplus v = w + v. \tag{14}$$

Also, notice that given x', since k_0 is random, v is random, and since x is random, w is random. These observations can be used to show that predicting $F_{(k_0,k_1)}(x)$

for a randomly selected input x based on $(x', F_{(k_0,k_1)}(x'))$ where F is defined by (9) is equivalent to (notice that k_1 is unknown and random) predicting z in (14) for a randomly selected input w and unknown/random v. This implies that the probability of SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$ returning 1 is equal to the probability of ZWGameAX$(\mathcal{A})$, see Algorithm 4, returning 1. This probability is maximized for $\mathcal{A}$ outputting a simulator $\mathcal{S}$ which on input w outputs a z that maximizes $|\{v : z \oplus v = v + w\}|$. In other words, z maximizes the number of collisions v that yield the same $z = (v+w) \oplus v$. In formula, $p[\mathcal{A}_0, \mathcal{A}_1]$ (where $\mathcal{A}_0$ and $\mathcal{A}_1$ are derived from $\mathcal{A}$ according to the transformation described above) is equal to

$$2^{-\lambda} \sum_w \max_z Prob_{v \leftarrow \{0,1\}^\lambda}[z \oplus v = v + w] \tag{15}$$
$$= 2^{-\lambda} \sum_w \max_z \frac{|\{v : z \oplus v = v + w\}|}{2^\lambda}.$$

Algorithm 4 Finding z based on w

function ZWGameAX$(\mathcal{A})$
  $v \in \{0,1\}^\lambda$ is a random input;
  $\mathcal{S} \leftarrow \mathcal{A}(1^\lambda)$;
  $w \in \{0,1\}^\lambda$ is a random input;
  if $\mathcal{S}(w) \oplus v = w + v$ **then**
    $b = 1$;
  else
    $b = 0$;
  end if
  Return b;
end function
/*On input w, an optimal $\mathcal{S}$ outputs a z which maximizes $|\{v : z \oplus v = v + w\}|$.*/

We will design an algorithm which given w finds the z which maximizes probability (15); this algorithm can be directly translated in an optimal simulator $\mathcal{S}$ in ZWGameAX$(\mathcal{A})$ and SoftwareUnclRespGame$(\mathcal{A}_0, \mathcal{A}_1)$.

Algorithm 5 computes for input (z, w) the set $\{v : z \oplus v = v + w\}$ represented as either the empty set $\emptyset$ or a string $v \in \{0, 1, *\}$ where $*$ can take on any bit value. So, the cardinality of set $\{v : z \oplus v = v + w\}$ is equal to 2^c where c is the number of $*$ symbols in the output v of the algorithm. The algorithm shortens the initial bit string w and z to smaller substrings of w and z of equal length throughout the recursive calls while producing a representation $v \in \{0, 1, *\}$.

The algorithm distinguishes between the following cases. First, if $|w| = |z| = 1$, then $w = \alpha$ and $z = \beta$ for some $\alpha, \beta \in \{0, 1\}$. In order to satisfy (14) $\beta \oplus v = \alpha + v$. Since $+$ is addition with carry with truncation, $\alpha + v = \alpha \oplus v$. Hence, $\{v : z \oplus v = w + v\} = \{0, 1\}$ which is represented by the string $*$.

Second, if $|w| = |z| \geq 2$, then $w = w'\alpha_1\alpha_0$ and $z = z'\beta_1\beta_0$ for some bit strings w' and z', and bits $\alpha_1, \alpha_0, \beta_1, \beta_0 \in \{0, 1\}$. Similarly, let $v = v'\gamma_1\gamma_0$. Then, $z \oplus v = v + w$ is equivalent to

$$\beta_0 \oplus \gamma_0 = \alpha_0 \oplus \gamma_0 \text{ and} \tag{16}$$

$$(z'\beta_1) \oplus (v'\gamma_1) = (w'\alpha_1) + (v'\gamma_1) + (0^{|w'|}(\alpha_0 \cdot \gamma_0)), \tag{17}$$

where $\alpha_0 \cdot \gamma_0$ represents bit multiplication and is equal to the carry of $\alpha_0 + \gamma_0$. Equation (16) is the projection of $z \oplus v = v + w$ on the first bit and equation (17) is the projection of $z \oplus v = v + w$ on the remaining bits. If $\beta_0 \neq \alpha_0$, then (16) cannot be satisfied and $\{v : z \oplus v = v + w\} = \emptyset$. If $\beta_0 = \alpha_0$, then we distinguish the cases $\beta_0 = 0$ and $\beta_0 = 1$.

Algorithm 5 Outputs the set $\{v : z \oplus v = v + w\}$

```
1: function SolutionSpace(z, w) /* defined for |z| = |w| ≥ 1 */
2:     if |w| = |z| = 1 then
3:         α = w; β = z;
4:         if α ≠ β then
5:             Return ∅;
6:         else
7:             Return ∗;
8:         end if
9:     else/* n = |w| = |z| ≥ 2 */
10:        w'α1α0 = w with α1, α0 ∈ {0, 1};
11:        z'β1β0 = w with β1, β0 ∈ {0, 1};
12:        if α0 ≠ β0 then Return ∅;
13:        else/* α0 = β0 */
14:            if β0 = 0 then
15:                z = z'β1; w = w'α1;
16:                v =SolutionSpace(z, w);
17:                Return v∗;
18:            else/*β0 = 1*/
19:                z = z'β1; w = w'α1 + 0^|w'|(β1 ⊕ α1);
20:                v =SolutionSpace(z, w);
21:                Return v(β1 ⊕ α1);
22:            end if
23:        end if
24:    end if
25: end function
```

For $\beta_0 = \alpha_0 = 0$, any γ_0 solves (16) and equation (17) simplifies to

$$(z'\beta_1) \oplus (v'\gamma_1) = (w'\alpha_1) + (v'\gamma_1).$$

This means that application of SolutionSpace$(z'\beta_1, w'\alpha_1)$ yields a representation $v \in \{0, 1, *\}$ of all the solutions for (17). Concatenating v with the wild

card $*$ (expressing that γ_0 can be any bit value) in line 17 gives a representation of $\{v : z \oplus v = v + w\}$.

For $\beta_0 = \alpha_0 = 1$, any γ_0 solves (16) and equation (17) simplifies to

$$(z'\beta_1) \oplus (v'\gamma_1) = (w'\alpha_1) + (v'\gamma_1) + (0^{|w'|}\gamma_0). \tag{18}$$

The projection of (18) is equivalent to the equation $\beta_1 \oplus \gamma_1 = \alpha_1 \oplus \gamma_1 \oplus \gamma_0$, i.e.,

$$\gamma_0 = \beta_1 \oplus \alpha_1.$$

Substituting this in (18) gives the equivalent equation

$$(z'\beta_1) \oplus (v'\gamma_1) = ((w'\alpha_1) + (0^{|w'|}(\beta_1 \oplus \alpha_1))) + (v'\gamma_1).$$

This means that the application of

$$\textsc{SolutionSpace}(z'\beta_1, (w'\alpha_1) + (0^{|w'|}(\beta_1 \oplus \alpha_1)))$$

yields a representation $v \in \{0, 1, *\}$ of all the solutions for (17). Concatenating v with $\gamma_0 = \beta_1 \oplus \alpha_1$ in line 21 gives a representation of $\{v : z \oplus v = v + w\}$.

As an invariant of algorithm 5 one can prove by induction in the length $\lambda = |z| \geq 1$ that, if $\{v : z \oplus v = v + w\} \neq \emptyset$, then

$$|\{v : z \oplus v = v + w\}| = 2^{\lambda - wt(\bar{z})}, \tag{19}$$

where $wt(\bar{z})$ is the Hamming weight of $\bar{z}$ and $z = \alpha'\bar{z}$ with $\alpha' \in \{0, 1\}$. Maximizing the probability in (15) given w is therefore equivalent to finding, given w, the minimal $wt(\bar{z})$ among all z for which $\{v : z \oplus v = v + w\} \neq \emptyset$.

Let $w = w'01^j0^a$ with $a \geq 0$ and $j \geq 1$. We distinguish two cases: $j = 1$ and $j \geq 2$. If $j = 1$, then $w = w'010^a$. By applying lines 12-17 a times and applying lines 18-19 the $(a + 1)$th time for $\alpha_0 = \beta_0 = 1$ and $\alpha_1 = 0$ shows that z must have the form $z = z'\beta_1 10^a$ otherwise no solution for $z \oplus v = v + w$ exists. The recursive call in line 20 uses a new $z = z'\beta_1$ and $w = w'\beta_1$. If $\beta_1 = 1$, then the new w equals $w = w'1$ and $wt(\bar{z}) = 2 + wt(\bar{z'})$ if $|w'| = |z'| \geq 1$, and $wt(\bar{z}) = 1$ if $|w'| = |z'| = 0$. If $\beta_0 = 0$, then the new w equals $w = w'0$ and $wt(\bar{z}) = 1 + wt(\bar{z'})$ if $|w'| = |z'| \geq 1$, and $wt(\bar{z}) = 1$ if $|w'| = |z'| = 0$.

If $j \geq 2$, then $w = w'01^i110^a$ for some $i \geq 0$. By applying lines 12-17 a times and applying lines 18-19 the $(a + 1)$th time for $\alpha_0 = \beta_0 = 1$ and $\alpha_1 = 1$ shows that z must have the form $z = z'\beta_1 10^a$ for some bit β_1 otherwise no solution for $z \oplus v = v + w$ exists. The recursive call in line 20 uses a new $z = z'\beta_1$ and a new $w = w'01^i1 + 0^{|w'|+1+i}(\beta_1 \oplus 1)$. If $\beta_1 = 1$, then the new w equals $w = w'01^i1$ and $wt(\bar{z}) = 2 + wt(\bar{z'})$ if $|w'| = |z'| \geq 1$, and $wt(\bar{z}) = 1$ if $|w'| = |z'| = 0$. If $\beta_0 = 0$, then the new w equals $w = w'10^{i+1}$ and $wt(\bar{z}) = 1 + wt(\bar{z'})$ if $|w'| = |z'| \geq 1$, and $wt(\bar{z}) = 1$ if $|w'| = |z'| = 0$.

For both cases, one can show that $wt(\bar{z})$ is minimized for $\beta_0 = 0$ and $z = z'010^a$. Given the above analysis the number of ones in $\bar{z}$ as a function $f(.)$ of w is equal to $f(w'01^j0^a) = 1 + f(w'0)$ for $j = 1$ and $f(w'01^j0^a) = 1 + f(w'10^{j-1})$ for $j \geq 2$. By using similar arguments we can also derive $f(1^j0^a) = 0$ for $j = 1$

and $f(1^j0^a) = 1$ for $j \geq 2$. Since $a \geq 0$ is arbitrary, we obtain the recursion $f(w'01^j0^a) = 1 + f(w')$ for $j = 1$ and $f(w'01^j0^a) = 1 + f(w'1)$ for $j \geq 2$. Since $f(w') \leq f(w'1)$ (the recursion shows that if $w' = w''1$ then $f(w') = f(w''1)$, and if $w' = w''0$ then $f(w') = f(w'') = f(w''01) - 1 = f(w'1) - 1$), we get the inequality

$$f(w'01^j0^a) \geq 1 + f(w') \text{ for } j \geq 1$$

with $f(1^j0^a) = 0$ for $j = 1$ and $f(1^j0^a) = 1$ for $j \geq 2$. So, the minimal weight of $\bar{z}$ as a function of w is at least the number of maximal length substrings 1^j0 with $j \geq 1$ in $\bar{w}$ where $w = \beta'\bar{w}$ with $\beta' \in \{0,1\}$. (Notice that the number of maximal length substrings 1^j0 with $j \geq 1$ in $\bar{w}$ is at most $\lambda/2$.) For completeness, algorithm 6 computes the z of minimal weight given w; this algorithm will be used by the adversary to construct a simulator.

Algorithm 6 Computing a $\bar{z}$ with $\{v : z \oplus v = v + w\} \neq \emptyset$ of minimal weight

```
1: function MinimalWeightSol(w)
2:     if w = 1^j 0^a for some j ≥ 1 and a ≥ 0 then
3:         if j = 1 then
4:             Return 0^a;
5:         else
6:             Return 0^{j-2} 10^a;
7:         end if
8:     end if
9:     if w = w'01^j 0^a for some j ≥ 1 and a ≥ 0 then
10:        if j = 1 then
11:            z =MinimalWeightSol(w');
12:        else
13:            z =MinimalWeightSol(w'1);
14:        end if
15:        Return z0^j 10^a;
16:    end if
17: end function
```

Let W_h, $h \leq \lambda/2$, be the number of bit strings $w = \beta\bar{w}$ with $\beta \in \{0,1\}$ and $\bar{w} \in \{0,1\}^{\lambda-1}$ of the form

$$\bar{w} = 0^{a_0}1^{1+a_1}0^{1+a_2} \ldots 1^{1+a_{2h-1}}0^{a_{2h}}$$

where $a_i \geq 0$ for all i. The previous analysis shows that for $w \in W_h$, $wt(\bar{z}) \geq h$. Together with (19) we obtain

$$2^{-\lambda} \sum_w \max_z \frac{|\{v : z \oplus v = v + w\}|)}{2^\lambda} \leq 2^{-\lambda} \sum_{h=0}^{\lambda/2} |W_h| 2^{-h}.$$

We notice that the mapping from $\bar{w}$ with $\beta\bar{w} \in W_h$ to

$$0^{a_0}10^{a_1}10^{a_2}1 \ldots 0^{a_{2h-1}}10^{a_{2h}} \in \{0,1\}^\lambda$$

is a bijective mapping to strings in $\{0,1\}^\lambda$ with exactly $2h$ ones. This proves $W_h = 2\binom{\lambda}{2h}$ (the factor 2 comes from the wild card β). Hence, (15) is upper bounded by

$$\begin{aligned}
&2^{-\lambda}\sum_{h=0}^{\lambda/2}|W_h|2^{-h} = 2^{-(\lambda-1)}\sum_{h=0}^{\lambda/2}\binom{\lambda}{2h}2^{-h} \\
&= 2^{-\lambda}\left(\sum_{h=0}^{\lambda}\binom{\lambda}{h}\sqrt{2}^{-h} + \sum_{h=0}^{\lambda}\binom{\lambda}{h}(-\sqrt{2})^{-h}\right) \\
&= \left(\frac{1+\frac{1}{\sqrt{2}}}{2}\right)^\lambda + \left(\frac{1-\frac{1}{\sqrt{2}}}{2}\right)^\lambda \approx 0.85^\lambda = 2^{-0.234\lambda}.
\end{aligned}$$

This can be interpreted as

$$p[\mathcal{A}_0, \mathcal{A}_1] \leq 2^{-0.234\lambda}.$$

The above derivation also proves that, for $0 \leq h \leq \lambda/2$, the number of w for which $\max_z Prob_{v\leftarrow\{0,1\}^\lambda}[z \oplus v = v + w] \leq 2^{-h}$ is at least equal to $\sum_{i=h}^{\lambda/2} W_i = 2^\lambda - \sum_{i=0}^{h-1} W_h = 2^\lambda - \sum_{i=0}^{h-1} 2\binom{\lambda}{2h}$. The probability that such a w is selected is equal to $\sum_{i=h}^{\lambda/2} W_i/2^\lambda$. This can be interpreted as

$$\alpha_h[\mathcal{A}_0, \mathcal{A}_1] \geq \sum_{i=h}^{\lambda/2} W_i/2^\lambda = 1 - (\sum_{i=0}^{h-1}\binom{\lambda}{2h})/2^{\lambda-1} \approx 1 - 2^{1-(1-H(2(h-1)/\lambda))\lambda},$$

(where $H(.)$ is the binary entropy function) which in turn proves

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \min_h \max\{2^{1-(1-H(2(h-1)/\lambda))\lambda}, 2^{-h}\}.$$

Let $\omega = 0.141$ be the solution of $1 - H(2\omega) = \omega$ and set $h = 1 + \lfloor\omega\lambda\rfloor$, then $2^{1-(1-H(2(h-1)/\lambda))\lambda} \leq 2^{1-(1-H(2\omega))\lambda} = 2^{1-\omega\lambda}$ and $2^{-h} \leq 2^{-\omega\lambda}$, hence,

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq 2 \cdot 2^{-\omega\lambda}.$$

This also shows that $\mathbf{a} \geq \omega$. Simulations with the optimal simulator derived from Algorithm 6 show that these bounds are tight.

C Worst-case security analysis S-Box-CBC

In order to prove Theorem 4, let us have a closer look at the derivation leading to (13). Either $\rho[S](w_{i+1}) = 2/2^m = 2^{-(m-1)}$ if $w_{i+1} \neq 0$, or $\rho[S](w_{i+1}) = 1$ if $w_{i+1} = 0$. This means that there are exactly $\binom{n}{j}(2^m-1)^{n-j}$ vectors w with exactly j different i for which $w_{i+1} = 0$. For these vectors w, $p[\mathcal{A}_0](w)$ (let $\mathcal{A}_0$

and $\mathcal{A}_1$ be derived from $\mathcal{A}$ according to the transformation described in Theorem 3) is equal to

$$\prod_{i=0}^{n-1} \rho[S](w_{i+1}) = 2^{-(m-1)(n-j)}.$$

Notice that $p[\mathcal{A}_0](w) \leq 2^{-h}$ for $(m-1)(n-j) \geq h$, i.e., $j \leq n - h/(m-1)$. This proves that, for $h \leq n(m-1)$,

$$\begin{aligned}\alpha_h[\mathcal{A}_0, \mathcal{A}_1] &= \sum_{j=0}^{\lfloor n-h/(m-1)\rfloor} \binom{n}{j} (2^m-1)^{n-j}/2^{\lambda} \\ &= 1 - \sum_{j=\lfloor n-h/(m-1)\rfloor+1}^{n} \binom{n}{j} (2^m-1)^{n-j}/2^{\lambda}.\end{aligned}$$

Below we will show that for $h = \lambda(m-1)^2/(m(2m-1))$,

$$1 - \alpha_h[\mathcal{A}_0, \mathcal{A}_1] \leq \frac{\lambda(m-1)}{m(2m-1)} 2^{-h}$$

and this shows that

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \max\{2^{-h}, \frac{\lambda(m-1)}{m(2m-1)} 2^{-h}\} = \frac{\lambda(m-1)}{m(2m-1)} 2^{-h}.$$

To finish the proof, we need to lower bound $\alpha_h[\mathcal{A}_0, \mathcal{A}_1]$. For $h \leq n(m-1)$,

$$\begin{aligned}\alpha_h[\mathcal{A}_0, \mathcal{A}_1] &= \sum_{j=0}^{\lfloor n-h/(m-1)\rfloor} \binom{n}{j} (2^m-1)^{n-j}/2^{\lambda} \\ &= 1 - \sum_{j=\lfloor n-h/(m-1)\rfloor+1}^{n} \binom{n}{j} (2^m-1)^{n-j}/2^{\lambda} \\ &\geq 1 - \sum_{j=\lfloor n-h/(m-1)\rfloor+1}^{n} \binom{n}{j} 2^{-mj} \\ &\geq 1 - \frac{h}{m-1} \cdot \max_{j=\lfloor n-h/(m-1)\rfloor+1}^{n} \binom{n}{j} 2^{-mj}.\end{aligned}$$

Notice that $\binom{n}{j}2^{-mj} \geq \binom{n}{j+1}2^{-m(j+1)} = \binom{n}{j}2^{-mj} \cdot 2^{-m}(n-j)/(j+1)$ for $(n-2^m)/(2^m+1) \leq j$. This inequality holds true if $j \geq n - h/(m-1)$ and $n - h/(m-1) \geq (n-2^m)/(2^m+1)$, or equivalently $(n+1)(m-1)2^m/(2^m+1) \geq h$. Notice that this is implied by $h \leq n(m-1)$ if and only if $2^m \geq n$. So, for

$$\begin{aligned}h \leq h' &= \min\{n(m-1), (n+1)(m-1)2^m/(2^m+1)\} \\ &= \begin{cases} (n+1)(m-1)2^m/(2^m+1), & \text{if } 2^m \leq n, \\ n(m-1), & \text{if } 2^m \geq n, \end{cases}\end{aligned}$$

we can prove by induction in j that

$$\max_{j=\lfloor n-h/(m-1)\rfloor+1}^{n} \binom{n}{j} 2^{-mj} \leq \binom{n}{\lfloor n-h/(m-1)\rfloor+1} 2^{-m(\lfloor n-h/(m-1)\rfloor+1)} \approx 2^{H\left(\frac{h}{n(m-1)}\right)n+h\frac{m}{m-1}-\lambda},$$

where $H(.)$ is the binary entropy function. Hence,

$$\alpha[\mathcal{A}_0, \mathcal{A}_1] \leq \min_{h\leq h'} \max\{2^{-h}, \frac{h}{m-1} 2^{H\left(\frac{h}{n(m-1)}\right)n+h\frac{m}{m-1}-\lambda}\}.$$

Notice that ideally we want both exponents to be equal, i.e.,

$$-h = H\left(\frac{h}{n(m-1)}\right) n + h\frac{m}{m-1} - \lambda,$$

then

$$h = \frac{m-1}{2m-1}\lambda - H\left(\frac{h}{n(m-1)}\right)\frac{m-1}{2m-1}n.$$

For $h = n(m-1)^2/(2m-1)$ which is $\leq h'$,

$$\begin{aligned}\alpha[\mathcal{A}_0, \mathcal{A}_1] &\leq \max\{2^{-n(m-1)^2/(2m-1)}, \frac{n(m-1)}{2m-1} 2^{n+nm(m-1)/(2m-1)-nm}\} \\ &= \max\{2^{-n(m-1)^2/(2m-1)}, \frac{n(m-1)}{2m-1} 2^{-(nm^2/(2m-1)-n)}\} \\ &= \frac{n(m-1)}{2m-1} 2^{-n(m-1)^2/(2m-1)} = \frac{\lambda(m-1)}{m(2m-1)} 2^{-\lambda(m-1)^2/(m(2m-1))}.\end{aligned}$$

Since the bounds used in the proofs do not have an assymptotic impact, we can conclude

$$\mathbf{a} = \frac{(m-1)^2}{m(2m-1)}.$$

D Hardware Implementation

In order to compare the hardware overhead of each scheme, we implemented ADD-XOR and S-BOX-CBC (for $m = 4$ and $m = 8$). We did not implement MULTIPLY-ADD because, according to our estimation, the lower bound of the gate equivalence is greater than that of the other schemes given a similar security level.

For a fair comparison, we assume that the NVM is byte addressable and only has one read port, which is a very commonly used NVM model. In addition, we assume that the format of input comes as we desire, such that the circuit does not need to store any input x that is not used in the current clock cycle. All the implemented functions are synthesized to 45nm technology NanGate library [24] using Synopsys Design Compiler [25].

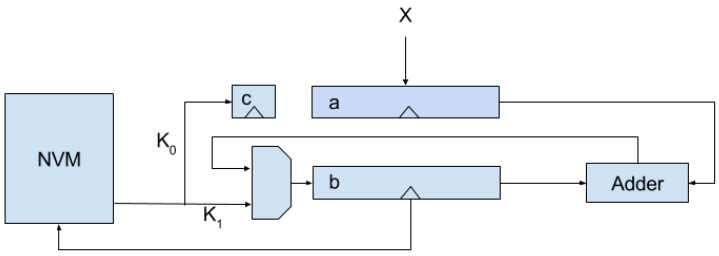

Figure 1. Hardware architecture of MULTIPLY-ADD.

D.1 MULTIPLY-ADD

In MULTIPLY-ADD, the integer multiplication is a heavy function to implement. Since $x \cdot k_0$ can be represented as $\sum_{i=0}^{\lambda}(x << i) \cdot k_{0,i}$, where $k_{0,i}$ represents the i-th bit of k_0, we suggest to use repeated addition and shifting to replace a full multiplier implementation.

The suggested hardware architecture is depicted in Figure 1. Register **a** stores the λ-bit value of x or the shifted value of x. Register **b** is first loaded with λ-bit k_1, then it is updated to the sum of the intermediate values in registers **b** and **a**. Register **c** is a one byte register, which loads one byte of k_0 to control whether the current content in register **a** should be added to register **b** or not. After λ rounds, register **b** will contain the final result of $x \cdot k_0 + k_1$.

To estimate the hardware overhead, we know that the implementation contains at least $2\lambda + 8$ bits of registers, λ full bit adders, and λ bit multiplexers to select loading data from NVM or the adder for register **b**. Given the conversion factors [3] from different gates to gate equivalence [24], we can conclude that a hardware implementation of the λ bit MULTIPLY-ADD function takes at least 22λ GE, which is more than the other candidates in our comparison.

D.2 ADD-XOR

The most efficient way of implementing ADD-XOR is to use one bit full adder, one XOR gate and one bit register for carry. But given the practical fact that the NVM is usually byte addressable, we have to read out one byte of key at once. We suggest to store 4 bits of k_0 and 4 bits of k_1 in one byte of NVM, such that we can fetch both k_0 and k_1 in one read operation, otherwise 8-bit registers will need to be be added in the circuit to store the intermediate value and we need to wait for k_1 to be fetched in the next clock cycle.

Figure 2 shows the hardware architecture of our implemented ADD-XOR function, which is extremely compact. It contains only a 4-bit full adder, 4 XOR

[3] 2-input NAND gate is 1.00 GE; 2-input AND gate is 1.33 GE; 2-input XOR gate is 2.00 GE; 1-bit register is 6.67 GE; 1-bit full adder is 6.33 GE; 2-to-1 multiplexer is 2.33 GE

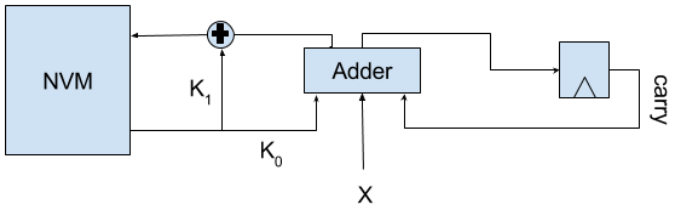

Figure 2. Hardware architecture of ADD-XOR.

gates and one bit register for the carry bit. The function itself takes only 64 gate equivalence[4]. This architecture does not require other registers to store intermediate values. As an optimal construction, all the values calculated in the current clock cycle are assumed to be written back to the NVM immediately.

D.3 S-BOX-CBC

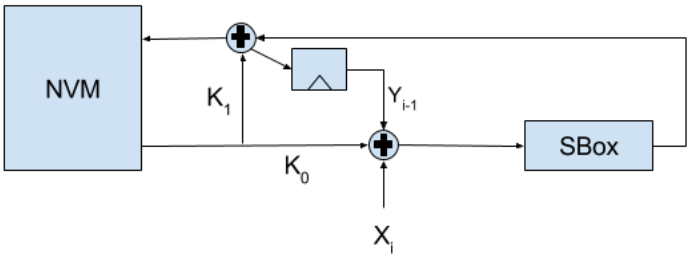

Figure 3. Hardware architecture of S-Box-CBC with $m = 4$. The S-Box is implemented as finite field multiplier circuit.

Our proposed S-Box takes as input $x \in GF(2^m)$ and outputs $x^3 \in GF(2^m)$. An implementation can come in two flavors which we discuss below:

D.3.1 Finite Field Multiplier We use finite field multipliers to implement x^3 in circuit. If latency would be a consideration, then we would use an orthonormal basis to represent elements in $GF(2^m)$ which makes squaring $x \to x^2$

[4] This gate equivalence number does not equal to the sum of different gates times the conversion factor, because this is the implementation result reported by the synthesis tool, which contains the area utilization of wire connections as well. This also applies to the gate equivalence number of the reported S-Box-CBC architectures.

a simple rotation operation leaving one more multiplication in order to compute $x^3 = x^2 \cdot x$. Since latency is not a consideration, we use one optimal circuit implementation for a finite field multiplier in $GF(2^m)$, which we reuse, in order to perform two multiplications $x^2 = x \cdot x$ and $x^3 = x^2 \cdot x$.

We apply Proposition 2, Equation (18), and Remark 1 in [26]: A multiplier in $GF(2^m)$ needs at most $3m$ 1-bit registers, $2m-1$ 2-input AND gates, and $(n+1)(m-1)+\omega-2-\sum_{i=1}^{n-1} r_i$ 2-input XOR gates, where n equals the number of nonzero entries in column 0 of the reduction matrix ($\mathbf{Q}$ in equation (8) in [26]), r_i is the position of the ith nonzero entry in this column, and ω equals the number of non-zero coefficients in the irreducible polynomial defining $GF(2^m)$.

For $m = 4$, we can use irreducible polynomial $x^4 + x + 1$ [27] with $n = 1$, and $\omega = 3$. This gives 12 1-bit registers, 7 AND gates, and 7 XOR gates. After conversion to gate equivalence, the finite field multiplication can be implemented in 103 GE.

The complete architecture of this scheme is in Figure 3. It contains one 4-bit S-Box, 8 XOR gates and 4 bit registers for storing y_{i-1}.

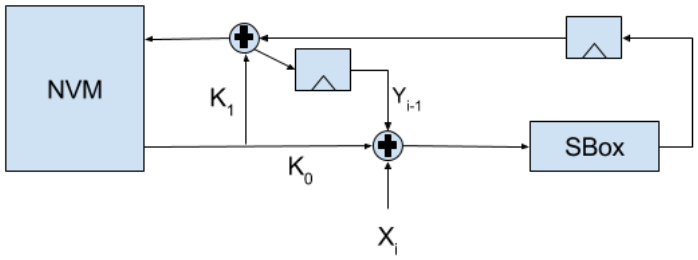

Figure 4. Hardware architecture of S-Box-CBC with $m = 8$. The S-Box is implemented as finite field multiplier circuit.

For $m = 8$, we can use irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ [27] with $n = 4$, $r_0 = 0$, $r_1 = 4$, $r_2 = 5$, $r_3 = 7$, and $\omega = 5$. This gives 24 1-bit registers, 15 AND gates, and 22 XOR gates. After conversion to gate equivalence, the finite field multiplication can be implemented in 224 GE.

The complete architecture of this parameter setting, see Figure 4, is slightly different from the one with $m = 4$. It requires one more register to store the output of the S-Box, because k_0 and k_1 cannot be fetched at the same clock cycle. Thus this architecture contains one 8-bit S-Box, 16 XOR gates and 16 bit registers for storing y_{i-1}.

D.3.2 Table Lookup We store a table of all the values x^3 and we implement the S-Box as a table lookup. The table has size $2^m \cdot m$ bits.

For $m = 8$, $2^m \cdot m$ equals 2048 bits. This means we need NVM to store these bits. However, the NVM storage required for storing key pairs and later

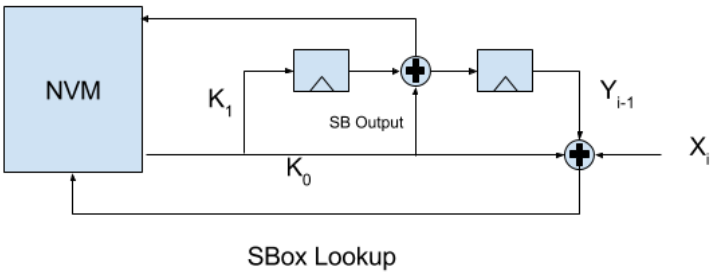

Figure 5. Hardware architecture of S-Box-CBC with $m = 4$. The S-Box is implemented as a lookup table in NVM.

reader events already takes 1360 bits for collision resistance 2^{-16} (see section 6). Together, this makes 3408 bits and makes a table lookup implementation impractical.

For $m = 4$, $2^m \cdot m$ equals only 64 bits. So we will need at least two NVM read operations to compute a 4-bit result. The implemented architecture is depicted in Figure 5. First, k_0 and k_1 are fetched from one byte, but k_1 needs to be stored in a 4-bit register, because it will be used at the next clock cycle. The input of the S-Box is computed as $y_{i-1} \oplus k_0 \oplus x_i$. This input is used as an address to fetch the S-Box output from the NVM. In the second clock cycle, the S-Box output is XORed with k_1 stored in the register. Also, 4 more bit of registers are required to store the value of y_i to compute y_{i+1}. We implemented this construction, and the total gate equivalence of this architecture is 518 GE and 557 GE for $\lambda = 60$ and 112, respectively. This is slightly less efficient than the architecture with an S-BOX implemented in circuit as we reported in Table 1.

D.4 Control Logic

In order to complete the comparison, we also implemented the control logic for our implemented functions. The control logic controls the implemented function and 40-bit one time pad, manages the memory interface and generates a signal to indicate the end of one reader event.

References

1. OECD/EUIPO, "Trade in counterfeit and pirated goods:mapping the economic impact." [Online]. Available: http://dx.doi.org/10.1787/9789264252653-en
2. H. Maleki, R. Rahaeimehr, C. Jin, and M. van Dijk, "New clone-detection approach for rfid-based supply chains," in *Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 122–127.
3. J. Shen, D. Choi, S. Moh, and I. Chung, "A novel anonymous rfid authentication protocol providing strong privacy and security," in *Multimedia Information Networking and Security (MINES)*. IEEE, 2010, pp. 584–588.

4. A. Ilic, M. Lehtonen, F. Michahelles, and E. Fleisch, "Synchronized secrets approach for rfid-enabled anti-counterfeiting," in *Demo at Internet of Things Conference*, 2008.
5. T. Dimitriou, "A lightweight rfid protocol to protect against traceability and cloning attacks," in *Security and Privacy for Emerging Areas in Communications Networks*. IEEE, 2005, pp. 59–66.
6. K. Bu, M. Xu, X. Liu, J. Luo, and S. Zhang, "Toward fast and deterministic clone detection for large anonymous rfid systems," in *Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2014, pp. 416–424.
7. C.-H. Hsu, S. Wang, D. Zhang, H.-C. Chu, and N. Lu, "Efficient identity authentication and encryption technique for high throughput rfid system," *Security and Communication Networks*, vol. 9, no. 15, pp. 2581–2591, 2016.
8. D. Zanetti, S. Capkun, and A. Juels, "Tailing rfid tags for clone detection," in *NDSS*, 2013.
9. D. Zanetti, L. Fellmann, S. Capkun *et al.*, "Privacy-preserving clone detection for rfid-enabled supply chains," in *RFID*. IEEE, 2010, pp. 37–44.
10. F. Kerschbaum and N. Oertel, "Privacy-preserving pattern matching for anomaly detection in rfid anti-counterfeiting." in *RFIDSec*. Springer, 2010, pp. 124–137.
11. R. Koh, E. W. Schuster, I. Chackrabarti, and A. Bellman, "Securing the pharmaceutical supply chain," *White Paper, Auto-ID Labs, Massachusetts Institute of Technology*, pp. 1–19, 2003.
12. G. EPCglobal, "Epc radio-frequency identity protocols generation-2 uhf rfid; specification for rfid air interface protocol for communications at 860 mhz–960 mhz," *EPCglobal Inc., November*, 2013.
13. A. Juels and S. A. Weis, "Authenticating pervasive devices with human protocols," in *CRYPTO05, LNCS 3621*. Springer, 2005.
14. G. J. Simmons, *Authentication Theory/Coding Theory*. Springer Berlin Heidelberg, 1985, pp. 411–431.
15. M. J. Dworkin, "Recommendation for block cipher modes of operation: The cmac mode for authentication," *Special Publication (NIST SP)-800-38B*, 2016.
16. J. Guo, T. Peyrin, and A. Poschmann, "The photon family of lightweight hash functions," *Advances in Cryptology–CRYPTO 2011*, pp. 222–239.
17. H. Maleki, R. Rahaeimehr, and M. van Dijk, "Sok: Rfid-based clone detection mechanisms for supply chains," in *Proceedings of the Workshop on Attacks and Solutions in Hardware Security(ASHES)*. ACM, 2017, pp. 33–41.
18. S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, and V. Khandelwal, "Design and implementation of puf-based unclonable rfid ics for anti-counterfeiting and security applications," in *RFID*. IEEE, 2008, pp. 58–64.
19. P. Tuyls and L. Batina, "Rfid-tags for anti-counterfeiting," in *Topics in cryptology–CT-RSA 2006*. Springer, pp. 115–131.
20. D. Ranasinghe, D. Engels, and P. Cole, "Security and privacy: Modest proposals for low-cost rfid systems," in *Auto-ID Labs Research Workshop*, 2004.
21. S. Tajik, E. Dietz, S. Frohmann, J.-P. Seifert, D. Nedospasov, C. Helfmeier, C. Boit, and H. Dittrich, "Physical characterization of arbiter pufs," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 493–509.
22. G. T. Becker, "The gap between promise and reality: On the insecurity of xor arbiter pufs," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 535–555.

23. U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 237–249.
24. NanGate, "Nangate open cell library - nangate," 2017. [Online]. Available: http://www.nangate.com
25. Synopsys, "Synopsys design compiler." [Online]. Available: https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html
26. A. Reyhani-Masoleh, "A new bit-serial architecture for field multiplication using polynomial bases," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008, pp. 300–314.
27. G. Seroussi, *Table of low-weight binary irreducible polynomials*, 1998.