A Leakage Resilient MAC

D. Martin, E. Oswald, M. Stam and M. Wójcik

Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom.
{dan.martin, elisabeth.oswald, martijn.stam}@bris.ac.uk

Abstract. We put forward the first practical message authentication code (MAC) which is provably secure against continuous leakage under the Only Computation Leaks Information (OCLI) assumption. We introduce a novel, modular proof technique: while most previous schemes are proven secure directly in the face of leakage, we reduce the (leakage) security of our scheme to its non-leakage security. This modularity, while known in other contexts, has two advantages: it makes it clearer which parts of the proof rely on which assumptions (i.e. whether a given assumption is needed for the leakage or the non-leakage security) and it also means that, if the security of the non-leakage version is improved, the security in the face of leakage is improved 'for free'. We feel that this is an advantageous proof technique, providing a better understanding of the scheme's security properties. In practice, we envisage that our scheme would be implemented using pairings on some pairing-friendly elliptic curve, where the 'leakiness' of the group operation can be experimentally estimated. We conclude the paper by discussing implementations; one on a popular core for embedded systems (the ARM Cortex-M4) and one on a high end processor (Intel i7), and investigate some performance and security aspects.

Keywords: Leakage Resilience, Message Authentication Code, Provable Security, Side Channels, Implementation

1 Introduction

Side channel leakage (*e.g.* via timing, power or EM side channels) enables the extraction of secret data out of cryptographic devices, as initially demonstrated by Kocher (*et al.*) in 1996 and 1999 [21, 22]. The engineering community reacted quickly by developing a variety of countermeasures that are commonly described as masking and hiding (see [23]). Such countermeasures intend to *reduce* the overall exploitable leakage via techniques that are cheap to implement.

Initially with hesitance, but more lately with much enthusiasm, the theory community picked up on the fact that schemes are needed which can tolerate some leakage. Complementary to the engineering approach, the aim is to design schemes which do not reduce leakage but cope with it, normally via updating the keys. The most compelling property of this approach is that the security definitions intrinsically incorporate leakage and hence security proofs then hold even in the presence of leakage. The main drawback of having theoretical backing of security seems to be that the resulting schemes are typically considerably less efficient than other schemes. A prime example of such a scheme is the stream cipher by Dziembowski and Pietrzak [8].

Despite the fact that almost all real word cryptographic protocols require some form of authentication, there is a distinct gap in the literature when it comes to leakage resilient message authentication codes (MACs). Hazay *et al.* [18] produce a MAC from minimal assumptions (existence of a one way function). While only relying on minimal assumptions is an advantage from a theoretical perspective, the scheme has a major drawback in that it only allows a bounded amount of leakage (this bound relates to the total leakage of the device). This makes the scheme unsuitable for practice. In his Master's thesis, Schipper [32] discusses a MAC construction in yet another security model. However unfortunately this MAC is also undesirable for practice as the number of AES calls used by verification grows logarithmically in the number of tag queries.

1.1 Our contribution

Inspired by the bilinear ElGamal cryptosystem by Kiltz and Pietrzak [19], we propose a MAC scheme that is secure within the continuous leakage model, using the Only Computation Leaks Information assumption (discussed in Sect. 2). To our knowledge this is the first MAC scheme to be given within this model, which has become one of the more desirable models due to its closer link with practical side channel scenarios.

In Sect. 3 we give our basic MAC construction and prove it secure in the random oracle model without leakage. Unlike previous work (where schemes have to be completely re-proven when considering leakage), we can construct our proof when considering leakage by a reduction to the non-leaky version (see Sect. 4). This is the first proof to achieve such a clean reduction, which has several advantages. Firstly it shows more clearly how much the leakage is impacting on the security of a scheme. This also implies if the security of the basic MAC construction is tightened, the security of the MAC construction with leakage is tightened 'for free'. This manifests itself (as seen in the theorem statement) by having the leakage security bound in terms of the security without leakage. Secondly it becomes clearer which further assumptions are required to prove security when assuming leakage: for example the basic MAC construction requires a Random Oracle assumption, while the Generic Group Model is required when leakage is added.

In Sect. 5 we discuss an implementation of our leakage resilient MAC when instantiated over a suitable, pairing supporting, elliptic curve using a well known library (MIRACL). We show that in practice (by compiling our implementation to two very different platforms, an embedded ARM core and a high end INTEL processor) we are reasonably efficient and the cost of providing provable leakage resilience, is not nearly as high as often believed. By inspecting power traces, we demonstrate that there are no (unforeseen) features that would weaken the implementation. In App. B we compare our MAC to the other leakage resilient MACs, as well as other schemes (*i.e.* PRFs, Signatures) which can be converted into MACs. We show that compared to the majority of other provably secure schemes we are considerably more efficient. The only scheme which is comparable with regards to efficiency is a signature scheme [16].

1.2 Related Work

Kiltz and Pietrzak [19] combine two techniques that are commonly used within both communities to build a key encapsulation mechanism on top of a key update scheme. The first technique is masking (or secret sharing as it is known by the theoretical community), which involves splitting the key into two parts and then working on each share separately. The second technique is frequent rekeying. Unlike other proposals (*e.g.* [20] or [1]), which are stateful (and thus need to be synchronised) or ones which needs to transmit a clue [25] to 'synchronise' parties,the proposal by Kiltz and Pietrzak [19] can leverage the algebraic properties of the underlying system such that the resulting sytem requires no synchronisation. This is achieved by changing the representation of the shares rather than changing the secret itself. Using the same techniques, Galindo and Vivek [16], and Tang *et al.* [35] create leakage resilient signature schemes. These constructions are proven secure in the continuous leakage model using the OCLI assumption [26](see also Sect. 2).

Albeit not related to goal of creating a MAC, there have been several recent papers which design leakage resilient schemes with the balance of provability and useability: schemes that come with some provable guarantee against arbitrary leaks without incurring prohibitively high overhead. When relaxing security notions from completely adaptive inputs (*i.e.* adversaries may choose input messages but also the side channel leakage adaptively) to non-adaptive security, simpler constructions for symmetric key cryptography can be achieved than previously thought [10]. In a differently motivated publication (proving existing schemes secure versus creating provably secure schemes), Balasch *et al.* [2] take a provably secure method, inner product masking, trim it down to implement a masked AES with it, and show this leads to a result which is comparable to other state of the art, yet not formally proven, masking approaches.

Dodis and Pietrzak [7] create a leakage resilient PRF where the leakage functions are chosen non-adaptively before any queries to the PRF are made. Faust *et al.* [10] construct a simpler leakage resilient PRF, which is acheived at the expense of having to make both the input to the PRF and the leakage non-adaptive. All known PRFs in the continual leakage model have the restriction of being non-adaptive (in the leakage), while MACs do not have this restriction. This shows of a seperation between PRFs and MACs which does not exist in the non-leakage model but PRFs will still serve as an interesting comparision.

2 Modelling Leakage

In this section we discuss what assumptions we make when modelling leakage. Clearly some restrictions are required on the leakage, otherwise the adversary will be able to win because he can just ask for the key. One of the first decisions to be made is how to define a bound for the leakage (*i.e.* how many bits about a secret does the adversary get via some side channel). For instance, one could define there to be an overall bound, *i.e.* the adversary gets at most a certain number of bits, irrespective of how often the construction is actually called (this is called bounded leakage in the literature). Another option would be to impose a per call bound. In this latter case, each call to the construction delivers at most a certain number of bits, while the overall leakage remains unbounded. This type of model is called continuous leakage model and fits best to real world leakage such as power or EM traces.

Whilst some previous works ([8, 11]) make an a priori assumption about the computational complexity of the leakage function, we opted for a concrete security statement. This means that the adversarial advantage is explicitly bounded in the complexity of the leakage function as expressed in the number of queries to the generic group oracles (see Sect. 2.2).

Finally we need to restrict the scope of the leakage function because otherwise (given our choices of assumptions above) no security would be possible (because of the infamous 'future computation attack' [19]). We discuss our choice of how to restrict the leakage function in the following.

2.1 Only Computation Leaks Information

Micali and Reyzin [26] introduced the Only Computation Leaks Information (OCLI) assumption. It states that data leakage only occurs on data that is currently being computed on and that data at rest will not leak. Whilst this assumption might not strictly hold in practice ([31] shows it to be invalid for some technologies on gate level), it sufficiently captures the behaviour of many state of the art devices.

Application of the OCLI assumption requires splitting a large computation into smaller components that each only operate on a subset of the data available, thus restricting the scope of what can be leaked on. OCLI will be modelled in this paper by splitting a function F into two parts $F^{◓}$ and $F^{◒}$. The part of the sensitive/exploitable input S used by $F^{◓}$ will be denoted $S^{◓}$ while the parts of the sensitive input used by $F^{◒}$ will be denoted $S^{◒}$. Without OCLI, a leakage query could potentially leak on both shares jointly, and thus reveal information about S. However due to OCLI, any leakage query can only ever leak on $S^{◓}$ and $S^{◒}$ independently, but never jointly on both.

Concretely, in our model the adversary may adaptively (per function call) choose leakage functions $l^{◓}, l^{◒}$ which will leak up to λ bits (this is a security parameter) on $F^{◓}$ and $F^{◒}$ respectively. The adversary also gets the output $l^{◓}(S^{◓}, x^{◓}, r^{◓})$ and $l^{◒}(S^{◒}, x^{◒}, r^{◒})$ where $x^{◓}, x^{◒}$ is the input to the functions and $r^{◓}, r^{◒}$ is the randomness that they use.

Note that while the leakage functions $l^{◓}$ and $l^{◒}$ can be chosen adaptively from query to query, they do have to be chosen at the same time for a single query. This mild restriction—that the leakage function $l^{◒}$ is not allowed to depend on the leakage obtained by $l^{◓}$—is quite common in the literature [16, 19], and reflects the abilities of a real world adversary (they can't change the measurement set-up mid measurement).

If this leakage process is iterated multiple times an index is used to specify which iteration we are on, for example we use $l_i^{◓}, l_i^{◒}, S_i^{◓}, S_i^{◒}, r_i^{◓}, r_i^{◒}$.

2.2 Bilinear Generic Group Model

We briefly recall the definition of bilinear groups and of bilinear maps, where we adhere to asymmetric pairings (see Galbraith *et al.* [13] for an overview). Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_3$ be cyclic groups all of prime order p with generators g_1, g_2, and g_3, respectively. A bilinear map is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ with the following properties; bilinearity states that $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z}_p : \; e(u^a, v^b) = e(u, v)^{ab}$, while non-degeneracy $e(g_1, g_2) \neq 1$, stops the construction of trivial maps. From this point onwards we define the generator g_3 of $\mathbb{G}_3$ to be $e(g_1, g_2)$.

The generic group model [24, 28, 34] is well established to prove the security of protocols involving elliptic curves. Its goal is to restrict the adversary in such a way that structure of the underlying group cannot be exploited (beyond what follows from the group axioms). This is achieved by representing each element within the group as a random string and providing oracles for the various group operations. As a consequence, given only a representation of a group element, the only ability the adversary has is to check equality (*i.e.* the adversary must use an oracle to perform any required group operations).

In the Generic Bilinear Group (GBG) model each of the three groups (or two when using a symmetric pairing) has its own randomised encoding. Each of these encodings will be represented by an injective encoding function $\xi_1 : \mathbb{Z}_p \rightarrow \Xi_1$, $\xi_2 : \mathbb{Z}_p \rightarrow \Xi_2$, $\xi_3 : \mathbb{Z}_p \rightarrow \Xi_3$ for $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ respectively, where Ξ_1, Ξ_2, Ξ_3 are sets of bitstrings. The adversary has access to the following 4 oracles:

- $\mathcal{O}_1(\xi_1(a), \xi_1(b)) = \xi_1(a + b \mod p)$
- $\mathcal{O}_2(\xi_2(a), \xi_2(b)) = \xi_2(a + b \mod p)$
- $\mathcal{O}_3(\xi_3(a), \xi_3(b)) = \xi_3(a + b \mod p)$
- $\mathcal{O}_e(\xi_1(a), \xi_2(b)) = \xi_3(a \cdot b \mod p)$

for all $a, b \in \mathbb{Z}_p$. Each of the 4 oracles will return $\perp$ if either of the inputs is not a invalid encoding of an underlying group element. $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ perform the group operations of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ respectively, while $\mathcal{O}_e$ performs the pairing operation. To work with these groups an adversary only needs to be given $\xi_1(1)$ and $\xi_2(1)$ (corresponding to the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively) plus access to the four oracles, from which any group element can be computed.

Leaking on generic group elements only reveals information about their representation. In some proofs (without leakage) that use the generic group model, the representation of group elements can be chosen in such a way that even sampling a random group element is hard (for an adversary). This is typically achieved by representing group elements as 'long' random strings. When leakage is included in proofs, such a strategy would not make sense because it would imply that only 'large' amounts of leakage[1] would strengthen the adversary. We instantiate the generic group model using compact representations instead. By setting $\Xi_i = \{0, 1\}^n$ where $n = \lceil \log p \rceil$ we get the unique representations required. This gives the adversary the ability to sample group elements efficiently and directly.

In contrast, Kiltz and Pietrzak [19] (and similarly, Galindo and Vivek [16]) use indirect sampling by raising some generator to a random exponent. They allow leakage on both the random representations,

[1] Typically one would need to leak significantly more than $\log p$ bits, where p would be the size of the group.

```
experiment $\mathbf{Exp}^{\mathrm{euf-cma}}_{\mathcal{M}}(A)$:
$K \xleftarrow{\$} KG()$
$S \leftarrow \{\}$
$(\sigma^*, m^*) \leftarrow A^{Tag(\cdot), Verify(\cdot,\cdot)}$
if $m^* \in S$ then
    return 0
end if
Return $VRFY(K, \sigma^*, m^*)$
```

```
proc $Tag(m)$:
$S \leftarrow S \cup \{m\}$
$\sigma \leftarrow TAG(K, m)$
Return $\sigma$
```

```
proc $Verify(\sigma, m)$:
$b \leftarrow VRFY(K, \sigma, m)$
Return $b$
```

Fig. 1: EUF-CMA experiment

```
proc $KG()$:
$K \xleftarrow{\$} \mathbb{G}_1$
Return $K$
```

```
proc $TAG(K, m)$:
$W \leftarrow H(m)$
$T \leftarrow e(K, W)$
Return $T$
```

```
proc $VRFY(K, T, m)$:
$W \leftarrow H(m)$
$T' \leftarrow e(K, W)$
Return $T' = T$
```

Fig. 2: Our bilinear MAC scheme $\mathcal{M}$

as well as their discrete logarithms (relative some generator). To model the adversary's ability to leak on the sampling computation itself. Our proof can be seen as more restrictive and our proofs only hold for implementing the sampling directly. We remark that it is possible to sample random elliptic curve points efficiently without performing an exponentiation with an unknown exponent. This is discussed in more detail in Sect. 5.

3 A MAC scheme

We define a MAC as a tuple of algorithms $\mathcal{M} = (KG, TAG, VRFY)$ such that:

$$\begin{aligned} K &\xleftarrow{\$} KG() \\ \sigma &\xleftarrow{\$} TAG(K, m) \\ b &\leftarrow VRFY(K, \sigma, m). \end{aligned}$$

For correctness we require for all valid keys K that $VRFY(K, TAG(K, m), m) = 1$. We use the standard definition of EUF-CMA security for the rest of this section, which is recapped below.

Definition 1 (Existential Unforgability Under Chosen Message Attack (EUF-CMA)). *Let $\mathcal{M} = (KG, TAG, VRFY)$ be a Message Authentication Code. Then Fig. 1 defines the EUF-CMA security game. The advantage of an adversary A winning the game is defined as* $\mathbf{Adv}^{\mathrm{eufcma}}_{\mathcal{M}}(A) = \Pr[\mathbf{Exp}^{\mathrm{eufcma}}_{\mathcal{M}}(A) = 1]$.

We now define our basic MAC construction. Using a hash function $H : \{0,1\}^* \times \{0,1\}^k \to \mathbb{G}_2$ our basic MAC scheme $\mathcal{M} = (KG, TAG, VRFY)$ is defined in Fig. 2. It can be shown to provide EUF-CMA security (Thm. 2). The scheme can be understood as follows; key generation consists of generating a random group element of $\mathbb{G}_1$. Tag generation first hashes the message, then takes the resulting hash as input to a bilinear map, using the secret key as other input. The MAC consists of a message, and its tag. Verification simply reconstructs the tag T and checks the correctness.

Before we provide the proof of the MAC we introduce a new Bilinear Diffie–Hellman problem, which we will use in the reduction to show the security of the MAC. This new DH problem will have its security 'sandwiched' between two other well known DH problems. We then introduce a variation of the problem, which makes the proof reduction slightly tidier but will have no effect on the security of the scheme.

3.1 A New Bilinear Diffie–Hellman Problem

In Definition 2 we introduce a bilinear problem, which we coin the target bilinear Diffie–Hellman (TBDH) problem. In Theorem 1 give a reduction to show if Co-Bilinear Diffie–Hellman (CBDH) is assumed to be a hard problem,[2] then so is the TBDH problem. Similarly, it can be shown that if the standard computational Diffie–Hellman (CDH) Problem is easy in $\mathbb{G}_3$ then the TBDH Problem is easy.

Definition 2 (Target Bilinear Diffie–Hellman Problem). *Given $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ with a bilinear map e between them, we say the Target Bilinear Diffie–Hellman (TBDH) Problem is hard if given g_2^x, g_3^y it is hard to compute g_3^{xy}, where x, y are sampled uniformly at random from $\mathbb{Z}_p$. Given an adversary A we define its advantage of winning this game as $\mathbf{Adv}^{tbdh}(A) = \Pr\left[\mathcal{A} = g_3^{xy} : \mathcal{A} \leftarrow A(g_1, g_2, g_2^x, g_3^y)\right]$.*

Before relating the TBDH problem to other Diffie–Hellman problems, we recall the CBDH Problem [36]. The CBDH problem states that given g_2^x, g_2^y, where x, y are sampled uniformly at random from $\mathbb{Z}_p$, you must find g_3^{xy}.

Theorem 1. *Let A be an adversary against the TBDH Problem, then there exists an adversary B (with approximately the same runtime as A) against the CBDH Problem, such that:*

$$\mathbf{Adv}^{\mathrm{tbdh}}(A) \leq \mathbf{Adv}^{\mathrm{cbdh}}(B) \ .$$

Proof. Let adversary A against TBDH be given, then adversary B that breaks CBDH is given in Fig. 3. From this we can see that B will win whenever A does and thus we have $\mathbf{Adv}^{tbdh}(A) \leq \mathbf{Adv}^{cbdh}(B)$. □

```
adversary $B(g_1, g_2, g_2^a, g_2^b)$:
$g_3^a \leftarrow e(g_1, g_2^a)$
$e(g_1, g_2)^{ab} \leftarrow A(g_1, g_2, g_2^b, g_3^a)$
Return $e(g_1, g_2)^{ab}$
```

Fig. 3: Constructing a CBDH Adversary from a TBDH Adversary

We now introduce the TBDHwO problem which will be used in the proof of security for the MAC. While it makes the reduction cleaner it does not weaken the security statement. The TBDHwO problem is the same as the TBDH problem (given g_2^x, g_3^y return g_3^{xy}) with an additional oracle $test(\cdot)$ which given an element checks if it is g_3^{xy}.

Definition 3 (Target Bilinear Diffie-Hellman with Oracle Problem). *We say the Target Bilinear Diffie-Hellman with Oracle (TBDHwO) Problem is hard if given g_2^x, g_3^y it is hard to compute g_3^{xy}, when given access to the $test(\cdot)$ oracle that checks if the given element is g_3^{xy}, where x, y are sampled uniformly at random from $\mathbb{Z}_q$. Given an adversary A we define its advantage of winning this game as $\mathbf{Adv}^{\mathrm{tbdhwo}}(A) =$* $\Pr[\mathcal{A} = g_3^{xy} : \mathcal{A} \leftarrow A^{test(\cdot)}(g_1, g_2, g_2^x, g_3^y)]$

Lemma 1. *Assuming the TBDH problem is hard then the TBDHwO problem is also hard. Moreover $\mathbf{Adv}^{\mathrm{tbdhwo}}(A) \leq (q_t + 1)\mathbf{Adv}^{\mathrm{tbdh}}(B)$, where q_t is the number of test queries made.*

[2] It is possible to modify our results to the usual notions of negligible advantages against probabilistic polynomial-time adversaries.

```
adversary $B(\mathcal{H} = g_2^x, \mathcal{F} = g_3^y)$:
$i \leftarrow 0$
$j \xleftarrow{\$} [q_h]$
$(T, , m) \leftarrow A^{H(\cdot), Tag(\cdot), VRFY(\cdot)}()$
Return $T$

simulator $H(m)$:
$i \leftarrow i + 1$
if $W[m] = \perp$ then
   if $i = j$ then
      $W[m] \leftarrow \times$
      Return $\mathcal{H}$
   else
      $W[m] \xleftarrow{\$} \mathbb{Z}_q$
   end if
end if
Return $h^{W[m]}$

simulator $Tag(m)$:
if $W[m] = \perp$ then
   $W[m] \xleftarrow{\$} \mathbb{Z}_q$
else if $W[m] = \times$ then
   $ABORT$
end if
$T \leftarrow \mathcal{F}^{W[m]}$
Return $T$

simulator $VRFY(T, m)$:
if $W[m] = \perp$ then
   $W[m] \xleftarrow{\$} \mathbb{Z}_q$
else if $W[m] = \times$ then
   Return $test(T)$
end if
Return $(T = \mathcal{F}^{W[m]})$
```

Fig. 4: Constructing a TBDHwO Adversary from a EUF-CMA Adversary

The proof of Lemma 1 is a standard hybrid argument: since every time A wants a *test* query answered, instead of asking the *test* oracle, he runs a copy of his algorithm and outputs the value he wants testing and based on whether this copy wins or not tells him what value the test function call will return. Since this needs to be done for each test function the bound given holds.

Theorem 2. *Let $H : \{0,1\}^* \times \{0,1\}^k \to \mathbb{G}_2$ be modelled as a random oracle and A be an EUF-CMA adversary against $\mathcal{M}$ who makes q_h queries to the hash function and q_v verification queries, then there exists an adversary B (of similar computational complexity) against the TBDH problem such that:*

$$\mathbf{Adv}_{\mathcal{M}}^{\text{eufcma}}(A) \leq (q_h + 1)(q_v + 1)\mathbf{Adv}^{\text{tbdh}}(B).$$

Proof. The proof works by reducing the problem of forging the MAC to the problem of solving the TBDH problem; let A be an adversary against the EUF-CMA security of $\mathcal{M}$, Fig. 4 shows the reduction on how an adversary B can solve the TBDHwO problem using the adversary A.

Without loss of generality we assume that, if A creates a forgery T on m, it has queried $H(m)$ (if it hasn't we can simply create an equivalent adversary that performs the same operations but hashes $H(m)$ before outputting the forgery, at the cost of one extra hash query). From this we can see that when the message forged on was the j^{th} query to the RO, B has won the TBDH game. The reduction constructs tags in such a way that it simulates having the key as g_1^x. If the adversary subsequently forges on a point whose hash is g_2^y, the resulting tag will be the answer to the TBDH problem (g_3^{xy}).

The probability of the message being forged on being the j^{th} RO call is $(q_h + 1)^{-1}$. The $ABORT$ does not affect this probability, since the only time B will abort is if A tries to tag on the value B wants it to make a forgery on and hence a forgery on this value is no longer possible. Thus $\mathbf{Adv}_{\mathcal{M}}^{\text{eufcma}}(A) \leq (q_h + 1)\mathbf{Adv}^{\text{tbdhwo}}(B)$ and by Lemma 1 the theorem holds (where q_t from Lemma 1 equals q_v). □

4 A Leakage Resilient MAC

We start this section by introducing the definition of a key update mechanism. Kiltz and Pietrzak [19] implicitly constructed and used a key update mechanism within their KEM. This key update mechanism was then used again in the signature scheme by Galindo and Vivek [16] and the signature scheme by Tang

```
proc $F^*(S_i^{◓}, S_i^{◒}, x)$:
$(S_{i+1}^{◓}, O) \xleftarrow{\$} F^{◓}(S_i^{◓}, x)$
$(S_{i+1}^{◒}, y) \xleftarrow{\$} F^{◒}(S_i^{◒}, O)$
Return $y$
```

Fig. 5: The algorithm F^*

et al. [35]. After showing that our definition aligns with the KP key update mechanism, we define what it means for a scheme to be compatible with a key update mechanism. We show this is the case for our MAC given in the previous section and then go on to prove our MAC secure in the face of leakage.

4.1 Key Update Mechanism

We define a key update mechanism as a set of tuples $\mathcal{KU} = (Share, Recombine, U^{◓}, U^{◒})$ such that:

$$\begin{aligned}(S_0^{◓}, S_0^{◒}) &\xleftarrow{\$} Share(K)\\ (S_{i+1}^{◓}, r_u) &\xleftarrow{\$} U^{◓}(S_i^{◓})\\ S_{i+1}^{◒} &\xleftarrow{\$} U^{◒}(S_i^{◒}, r_u)\\ K_i &\leftarrow Recombine(S_i^{◓}, S_i^{◒})\end{aligned}$$

For correctness we require that $Recombine(Share(K)) = K$.

We define an equivalence class as follows; we say $(S_i^{◓}, S_i^{◒}) \equiv (S_j^{◓}, S_j^{◒})$ if $Recombine(S_i^{◓}, S_i^{◒}) = Recombine(S_j^{◓}, S_j^{◒})$. Then the final requirement is that the algorithms $U^{◓}, U^{◒}$ preserve the equivalence class of the shares (and thus $\forall i : K_i = K$). Formally we require $(S_i^{◓}, S_i^{◒}) \equiv (S_{i+1}^{◓}, S_{i+1}^{◒})$ where $(S_{i+1}^{◓}, O_i) \xleftarrow{\$} U^{◓}(S_i^{◓}), S_i^{◒} \xleftarrow{\$} U^{◒}(S_i^{◒}, O_i)$.

The KP key update mechanism used within the KEM [19] can be seen to fit within this framework. This is due to the fact that the key is initially split into two shares which multiply together to give back the original key. The first share is updated by multiplying it by a random value, while the second share is updated by multiplying it by the inverse of the random value. This forms our equivalence class and thus when the two shares are multiplied together we will recover the orignal key, regardlesss of how many times the shares have been updated. The KP key update mechanism will be used for the remainder of this paper (and denoted $\mathcal{KU}$).

Definition 4 (Key Update Splittable). *We say that a tuple of functions $(F^{◓}, F^{◒})$ is a split of F conforming to key update mechanism $\mathcal{KU}$ if the following two properties hold. Firstly:*

$$\{F(K, x)\}_{\mathcal{R}} = \{F^*(Share(K), x)\}_{\mathcal{R}^*}$$

where F^* is defined in Fig. 5, the equivalence is over the randomness from sets $\mathcal{R}, \mathcal{R}^*$ used by F, F^* respectively. Secondly, that for all sharings $(S_0^{◓}, S_0^{◒})$ the joint distribution on $(S_1^{◓}, S_1^{◒})$ after F^* has been called once is the same as if $(S_0^{◓}, S_0^{◒})$ had been updated using $(U^{◓}, U^{◒})$.

Claim. The MAC $\mathcal{M}$ given in Sec. 3 is Key Update Splittable conforming to the KP Key Update Mechanism $\mathcal{KU}$.

```
proc $KG()$:
    $K \xleftarrow{\$} \mathbb{G}_1$
    $S_0^{◓} \xleftarrow{\$} \mathbb{G}_1$
    $S_0^{◒} \leftarrow K \cdot (S_0^{◓})^{-1}$
    Return $(S_0^{◓}, S_0^{◒})$

proc $TAG^{◓}(S_i^{◓}, m)$:
    $W \leftarrow H(m)$
    $t_i^{◓} \leftarrow e(S_i^{◓}, W)$
    $r_{i+1} \xleftarrow{\$} \mathbb{G}_1$
    $s_{i+1}^{◓} \leftarrow S_i^{◓} \cdot r_{i+1}$
    Return $(S_{i+1}^{◓}, r_{i+1}, t_i^{◓}, W, w)$

proc $TAG^{◒}(S_i^{◒}, t_i^{◓}, W, w, r_{i+1})$:
    $t_i^{◒} \leftarrow e(S_i^{◒}, W)$
    $S_{i+1}^{◒} \leftarrow S_i^{◒} \cdot r_{i+1}^{-1}$
    $T \leftarrow t_i^{◓} \cdot t_i^{◒}$
    Return $(S_{i+1}^{◒}, (T, w))$

proc $VRFY(K, (T, w), m)$:
    $W \leftarrow H(m, w)$
    $T' \leftarrow e(K, W)$
    Return $(T' = T)$
```

Fig. 6: Leakage Resilient MAC $\mathcal{M}^*$

Proof. $\mathcal{M}$ can be converted into $\mathcal{M}^*$ which is given in Fig. 6.

Since we have that:

$$\begin{aligned} Tag^*(S_i^{◓}, S_i^{◒}, m) &= T \\ &= t^{◓} \cdot t^{◒} \\ &= e(S_i^{◓}, H(m)) \cdot e(S_i^{◓}, H(m,)) \\ &= e(S_i^{◓} \cdot S_i^{◒}, H(m, w)) \\ &= e(K, H(m, w)) \\ &= Tag(K, m). \end{aligned}$$

Hence $\mathcal{M}$ is Key Update Splittable □

There are three algorithms in our leakage resilient MAC: Key Generation, Tag and Verify. Our security definition only allows to leak on Tag, and we now explain why this is necessary. The Key Generation must not leak because it would leak on the original key. In practice, typical (security) devices would be shipped with their keys preinstalled and only the update would be done on the device. This leaves us to consider whether Tag (EUF-CMA-LT) or Verify (EUF-CMA-LV), or both (EUF-CMA-LTV) are allowed to leak. This question has not been considered before in the continual poly-time leakage model in the case of symmetric schemes, as all previous schemes in this model were public-key in which the question simply does not arise.

Making Verify leaky is problematic, because we are allowing adaptive leakage: assume the adversary takes a random group element and a message and sends both to Verify. In our construction (which follows a typical design) Verify has to calculate the correct tag first, and then compare it against the submitted tag. Hence the adversary can keep submitting the same message until he has completely leaked the tag created for comparison. This tag can then be submitted as a forgery since it was never requested from the Tag oracle. This attack will work against any MAC construction which requires a reconstruction of the TAG as part of Verify and is hence not specific to our MAC. We leave it as a question for future research how to deal with a leaky Verify theoretically. In practice, Verify will leak and whilst we cannot formally include it in the security proof, we can assume that practical countermeasures can be put in place.

Definition 5 (Existential Unforgability Under Chosen Message Attack with Tag Leakage (EUF-CMA-LT)). *Let $\mathcal{M}^* = (\mathcal{KU}, TAG^{◓}, TAG^{◒}, VRFY)$ be a Message Authentication Code. Then*

```
experiment $\mathbf{Exp}^{\mathrm{eufcmalt}}_{\mathcal{M}}(A)$:
$K \overset{\$}{\leftarrow} KG()$
$(S^{◓}_0, S^{◒}_0) \overset{\$}{\leftarrow} SHARE(K)$
$S \leftarrow \{\}$
$(\sigma^*, m^*) \leftarrow A^{Tag(\cdot),Verify(\cdot,\cdot)}$
if $m^* \in S$ then
    return 0
end if
Return $VRFY(K, \sigma^*, m^*)$
```

```
proc $Tag(m, l^{◓}_i, l^{◒}_i)$:
$S \leftarrow S \cup \{m\}$
$(S^{◓}_{i+1}, O_i) \xleftarrow{r^{◓}_i} TAG^{◓}(S^{◓}_i, m)$
$\Lambda^{◓}_i \leftarrow l^{◓}_i(S^{◓}_i, r^{◓}_i)$
$(S^{◒}_{i+1}, \sigma) \xleftarrow{r^{◒}_i} TAG^{◒}(S^{◒}_i, O_i)$
$\Lambda^{◒}_i \leftarrow l^{◒}_i(S^{◒}_i, r^{◒}_i, O_i)$
Return $(\sigma, \Lambda^{◓}_i, \Lambda^{◒}_i)$
```

```
proc $Verify(\sigma, m)$:
$b \leftarrow VRFYK, \sigma, m$
Return $b$
```

Fig. 7: EUF-CMA-LT experiment

Fig. 7 defines the EUF-CMA-LT security game. The advantage of an adversary A winning the game is defined as $\mathbf{Adv}^{\mathrm{eufcmalt}}_{\mathcal{M}}(A) = \Pr[\mathbf{Exp}^{\mathrm{eufcmalt}}_{\mathcal{M}}(A) = 1]$.

Theorem 3. *The MAC $\mathcal{M}^*$ is EUF-CMA-LT secure in the Generic Group Model. The advantage of a q-query (to the generic group oracles) adversary who is allowed λ bits of leakage is given by:*

$$\mathbf{Adv}^{\mathrm{eufcmalt}}_{\mathcal{M}^*}(A) \leq 2^{4\cdot\lambda} \cdot \mathbf{Adv}^{\mathrm{eufcma}}_{\mathcal{M}}(B) + \frac{q^2}{p}.$$

Proof. This proof is given in the Generic Group Model and shows that even with the use of leakage the adversary cannot get any elements that they could not get when no leakage was involved. After this has been shown it is reasonably straightforward to argue that without learning any new elements from the leakage then the leakage can increase the adversary's advantage by at most the number of bits that is leaked on for a single element. By showing that each element is only leaked at most four times, we get that the advantage can only be increased by at most $2^{4\cdot\lambda}$ over the advantage in the game where no leakage is involved.

We will represent group elements with polynomials, which will be instantiated at the end of the computation. The polynomials allow the game to keep track of which elements the adversary has asked for in a straightforward manner and because they are instantiated at the end, the adversary's decisions clearly can not be dependant on the actual values of the elements. Instatiation of the polynomials at the end is a common trick used within the litrature but means that if two (non equal) polynomials, when instantiated, collide the simulation fails as a single group element now has multiple representations. Thus we must also show that the chance of an adversary forcing this collision is also small.

Let $K, \{R_i\}^{q_T}_{i=0}, \{H_i\}^{q_H}_{i=1}, \{U_i\}^{2q_O}_{i=1}, \{V_i\}^{2q_O}_{i=1}, \{W_i\}^{2q_O}_{i=1}$ be indeterminants where q_H is the number of hash queries, q_T is the number of Tag calls and q_O is the number of group oracle calls (let $q = q_H + q_T + 3 \cdot q_O$). The indeterminants represent the following; K is the secret key, $\{R_i\}^{q_T}_{i=0}$ are the randomness used to update the key, $\{H_i\}^{q_H}_{i=1}$ represent any hash function queries and $\{U_i\}^{2q_O}_{i=1}, \{V_i\}^{2q_O}_{i=1}, \{W_i\}^{q_O}_{i=1}$ represent any elements that are guessed in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ respectively. The lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are used to keep track of polynomials and their representations in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ respectively. They are initialised as follows:

$$\mathcal{L}_1 = \{(1, \xi^1_1\} \cup \{(R_i, \xi^1_{i+2})\}^{q_T}_{i=0}$$
$$\mathcal{L}_2 = \{(1, \xi^2_1)\}$$
$$\mathcal{L}_3 = \{(1, \xi^3_1)\}$$

All three lists are initially instantiated with the identity, note that it is not strictly necessary to instantiate the identity in $\mathbb{G}_3$ since it can be calculated. We precompute the representations of the randomness but since the adversary does not have access to this list of elements (and instantiation, defining the element happens at the end) this does not change the game but makes it notationally simpler. All of the representations in this step are chosen randomly with the requirement that they are unique.

The oracles used by the Generic Group model are given in Fig. 8 in App. A.

The Adversary A outputs m, T and is said to have won if:

1. $F_i^l = F_j^l$ for $l \in \{1, 2, 3\}$ and $i \neq j$
2. $K \cdot H^* - T = 0$ where H^* is the indeterminant corresponding to the hash of m

The first case corresponds to the adversary being able to create two polynomials which evaluate to the same value. The second case corresponds to the adversary being able to create a forgery on the MAC.

Since all polynomials in the original lists are degree one and the only operation that increases the degree is the pairing operation which can only be called on elements in $\mathbb{G}_1, \mathbb{G}_2$. This means that we can have degree two polynomials in $\mathbb{G}_3$ but not the other two groups. Hence by the Swartz-Zippel lemma we have that the probaility of two (non-zero) polynomials evaluate to the same value is $\frac{2}{p}$. Since there are at most q polynomials we get that there are $\binom{q}{2} \leq \frac{q^2}{2}$ pairs of polynomials that could collide and thus the probability of any two polynomials colliding is $\frac{q^2}{p}$.

Without loss of generality, we will now only look at leakage in the target group $\mathbb{G}_3$ since any element from $\mathbb{G}_1, \mathbb{G}_2$ calculated by the leakage can be transfered over to $\mathbb{G}_3$ using a pairing and any elements known to the adversary can easily be recomputed within the leakage.

While we now only need to look at leakage in the target group since each leakage function has access to different secret elements, the set of elements that can be calculated by each will be different and thus must be considered individually. The adversary will win if he can repeatedly get a (new) Tag into the leakage function to reveal it, or he can cause collisions within the leakage. If he can only create a forgery once he will not be able to leak the complete tag and thus he is required to get in into multiple leakage sets

Let $L_i^{◓}$ be the set of elements that could be computed by the leakage function $l_i^{◓}$, then:

$$L_i^{◓} = \{A \cdot S_i^{◓} + B \cdot R_{i+1} + C\}$$

Where $A, B \in \mathbb{F}_p[\{H_i\}_{i=0}^{q_H}, \{V_i\}_{i=0}^{2q_O}]$ and $C \in \mathbb{F}_p[K\{H_i\}_{i=0}^{q_H}, \{U_i\}_{i=0}^{2q_O}, \{V_i\}_{i=0}^{2q_O}, \{W_i\}_{i=0}^{q_O}]$ and we use $S_i^{◓}$ to denote $\sum_{j=0}^{i} R_i$.

Let $L_i^{◒}$ be the set of elements that could be computed by the leakage function $l_i^{◒}$, then:

$$L_i^{◒} = \{A \cdot S_i^{◒} + B \cdot R_{i+1} + C + d \cdot S_i^{◓} \cdot H_i\}$$

Without loss of generality we will assume that ith tag call maps to H_i. Also we have that $d \in \mathbb{F}_p$ and $S_i^{◒}$ denotes $K - \sum_{j=0}^{i} R_i$.

The only Tags that can be contained in $L_i^{◓}$ are tags of the form $H_j \cdot K$ for $j < i$ in which the adversary has included the tag via F and thus there is no advantage in leaking upon $H_j \cdot K$. Ignoring this trivial case, there are no linear combinations possible that will reveal an unknown tag or the key itself. Similarly for $L_i^{◓}$ the only tags that can be included are of the form $H_j \cdot K$ for $j \leq i$ this is because he can again ask to leak on tags he has already seen by embedding them in F but this time can also leak on $H_i \cdot K$ but this is also not of any use because the leakage on this tag will be received at the same time the adversary is given the Tag and thus no extra information is gained.

From this point on we will only consider leakage on elements that contain an unknown component, since if all components are completely known by the adversary it is not worth learning leakage on. All that is left to show is that no element can be leakaged upon more than a bounded number of times (we can show this to be 4 times):

- $S_i^{◓}$ can be leaked on twice $L_{i-1}^{◓}, L_i^{◓}$
- $S_i^{◒}$ can be leaked on twice $L_{i-1}^{◒}, L_i^{◒}$
- R_{i+1} can be leaked on twice $L_i^{◓}, L_i^{◒}$
- $S_i^{◓} \cdot H_i$, the intermediate state, can be leaked on 4 times $L_{i-1}^{◓}, L_i^{◓}, L_{i-1}^{◒}, L_i^{◒}$

Since each element can only be leaked on at most four times, the adversary can only learn up to $4 \cdot \lambda$ bits of information per secret. Thus we get the bound as stated in the theorem, since after collisions the adversary's advantage can be at most $2^{4 \cdot \lambda}$ times the advantage of playing the standard non-leakage game. $\square$

5 Practical Aspects of our Scheme

For a practical implementation we selected the Barreto-Naehrig (BN) [3] family of pairing-friendly curves. The family of BN curves are defined over a prime field $\mathbb{F}_q$, with prime order and are given by the equation $E : y^2 = x^3 + b$, with $b \neq 0$ (we select $b = 2$). The common feature of this family is their embedding degree of $k = 12$, which to some extent, dictates the security level achieved on the curve. For the implementation purpose we focused on a security level equivalent to 128-bit and 192-bit AES, for which BN curves are ideally suited.

The prime q is given by polynomial $q = 36u^4 + 36u^3 + 24u^2 + 6u + 1$. For efficiency we set $u = -(2^{62} + 2^{55} + 1)$ and $u = -(2^{190} + 2^{19} + 2^{17} + 2^{15} + 2^{13} + 2^{12} + 2^{11} + 2^9 + 2^8 + 2^7 + 2^5 + 2^4 + 2^3 + 1)$ for 128 and 192 bits security level respectively [30]. That determines the size of the operands in the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$, which are over $\mathbb{F}_q$, $\mathbb{F}_{q^2}$ and $\mathbb{F}_{q^{12}}$ respectively. In case of the 128 bit security level, operations are carried out on operands of length 254, 508 and 3048 bits, whereas for 192 bits security level are carried out on operands of 766, 1532 and 9192 bits.

All algorithms of our schemes were implemented using the MIRACL software library [33], which is a portable C/C++ library that supports a wide-rage of different platforms including embedded ones. The advantage of the selected library is the extensive support for highly efficient pairing operations. In addition we extended and adopted functionality provided by the library to our particular case, whenever required. For the underlying pairing operations, MIRACL uses the well-known Miller algorithm [27]. Furthermore, it also applies the Galbraith-Scott method [14], which allows computing homomorphisms between the groups $\mathbb{G}_2$ and $\mathbb{G}_3$ efficiently. These homomorphisms are further used to speed up arithmetic computations in $\mathbb{G}_2$ and $\mathbb{G}_3$ by applying Gallant-Lambert-Vanstone method [17] (which works when a suitable group homomorphism is given). All mentioned optimisation strategies increase efficiency of pairing computations, thus speeding up our proposed scheme and making it also suitable for more resource-constrained environments.

We implemented and measured execution times of the schemes on both an embedded platform and a high-end device. For the former case, as a target platform, we selected a popular STM32F4Discovery board, which houses 32-bit ARM Cortex-M4 CPU. For the latter case, we utilised the 64-bit Intel Core i7 CPU. The internal clock of the Cortex-M4 was set to available maximum, *i.e.*, 168MHz, whereas the Intel i7 ran with a 3GHz clock. We ran our benchmarks several times to derive median timings for inclusion in this paper.

Table 1: Performance comparison of random point generation methods.

Device	Cortex-M4		Intel i7	
Operation	Time (ms)		Time (μs)	
	128-bits	192-bits	128-bits	192-bits
Random_Sampler	36	588	96	1159
Try_and_Increment	34	569	76	1119
Random_Scalar	173	2827	389	5186
SWEncoding	30	616	121	1217

5.1 Generating random curve points

Generating random group elements securely is vital for key generation and key updates (which happen in TAG and Verify). We found four options (see Fig. 9 for algorithmic descriptions) for this purpose, which we now discuss in turn. The first one, the Random_Sampler procedure, randomly selects an x-coordinate and checks if it is on the curve. In case of success, the procedure computes an associated y-coordinate, otherwise randomly selects another x-coordinate. The second key generation procedure, Try_and_Increment is very similar to the first one. It differs only in re-selection of x-coordinate in which the procedure increments x-coordinate by 1 and repeats the assessment of whether a new x is on the curve. The third one, Random_Scalar selects a scalar at random and preform scalar multiplication using a fixed group generator, *i.e.*, $P = (-1, 1)$. The last procedure uses the encoding to BN curve [12], where a random element $t \in \mathbb{F}_q$ is transformed into an element of the curve $E(\mathbb{F}_q)$, which was used in [15]. Note that when using this encoding one has to perform it twice in order to generate a point distributed uniformly at random [15]. Hence in practice the timings are effectively twice as long.

Performance For a fair comparison of timings, all procedures were implemented without blinding. Applying blinding to the Random_Sampler, Try_and_Increment and Random_Scalar methods requires one additional multiplication. A more involved blinding method for the BN encoding have been proposed in [12]. Table 1 shows that Random_Sampler and Try_and_Increment are by far the most efficient (recall that the SWEncoding method needs to be performed twice for uniformly distributed points), and it is clear that this advantage would also hold when blinding is included. This is good news as the Random_Scalar method is not only slow, but also known to be very vulnerable to power analysis attacks [6].

Attack vectors It has been shown practically that the Random_Scalar method can completely leak the entire secret randomness and hence strictly speaking, it cannot be used for schemes proved secure in the continual leakage model. Security aspects of the SWEncoding scheme were discussed in [15]. They conclude that the SWEncoding might leak via the Jacobi symbol. The security of the other two methods w.r.t. the continual leakage model has not yet been investigated. Hence we will now discuss the security considerations here.

The Try_and_Increment procedure will leak information about the number of increments via its overall execution time (which will be visible from power or EM traces). It is not obvious how this information could be utilised efficiently. However it will contribute to the amount of leakage per call for the λ security bound.

The Random_Sampler method chooses values for x independently of previous choices and hence does not leak any additional information on the x from its high level functionality. The only part which may reveal information about the point is the calculation of the Jacobi symbol.

Since for both the SWEncoding and the Random_Sampler any leakage will be from the Jacobi Symbol we now discuss the leakage that may be available during its computation. The Jacobi computation has

Table 2: Performance comparison

(a) Bilinear MAC

Device	Cortex-M4		Intel i7	
Operation	Time (ms)		Time (μs)	
	128-bits	192-bits	128-bits	192-bits
TAG	2146	30317	7935	68692
$VRFY$	2146	30317	7958	68687

(b) LR MAC

Device	Cortex-M4		Intel i7	
Operation	Time (ms)		Time (μs)	
	128-bits	192-bits	128-bits	192-bits
KG^*	72	1126	170	2128
TAG^*	4059	57274	15473	130612
$VRFY^* = VRFY$	2146	30317	7958	68687
$Share$	34	566	94	1082
$TAG^{◓}$	2183	30883	7974	69650
$TAG^{◒}$	1874	26382	7162	60841
$Recombine$	2	11	< 1	< 1

a conditional operation which swaps the numerator and denominator when certain conditions are met. We recorded power traces on the embedded platform, see Fig. 10 for the comparison for two different inputs which have 99 and 96 conditional switches respectively. While it is not clear how much use this information is or how it can be exploited, it is recommended to use blinding (with r^2) to hide the point since if $x^3 + b$ is square, $r^2(x^3 + b)$ will also be square for all random r. The advantage here is that the Try_and_Increment can not as easily be blinded as the Jacobi and thus this is preferable.

5.2 Performance of the overall scheme

Finally we give an overview of the performance of the high level functions of the MAC constructions. The basic MAC scheme had identical operations in is TAG and VRFY procedures (bar the additional equality check in VRFY which is extremely fast), hence the resulting identical timings.

Switching then to the leakage resilient version, it is clear that the cost essentially doubles for the TAG computation. Since we had to assume VRFY was not leaking (recall that our construction, like other MAC constructions requires the reconstruction of TAG during VRFY, which is seemingly impossible to do securely allowing adaptive adversaries in the continual leakage model), VRFY was not defined (and hence not implemented) in a secret shared manner. Hence the timings for VRFY are the same as for the bilinear MAC construction.

Acknowledgements

Dan Martin and Elisabeth Oswald have been supported in part by EPSRC via grant EP/I005226/1. Marcin Wójcik has been supported by the EU DG Home Affairs - ISEC (Prevention of and Fight against Crime) / INT (Illegal Use of Internet) programme and his research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^o 609094.

References

1. Abdalla, M., Belaïd, S., Fouque, P.A.: Leakage-resilient symmetric encryption via re-keying. In: Bertoni, G., Coron, J.S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 471–488. Springer, Berlin, Germany, Santa Barbara, California, US (Aug 20–23, 2013)

2. Balasch, J., Faust, S., Gierlichs, B., Verbauwhede, I.: Theory and practice of a leakage resilient masking scheme. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 758–775. Springer, Berlin, Germany, Beijing, China (Dec 2–6, 2012)
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Berlin, Germany, Kingston, Ontario, Canada (Aug 11–12, 2005)
4. Blömer, J., Günther, P., Liske, G.: Improved side channel attacks on pairing based cryptography. Cryptology ePrint Archive, Report 2011/706 (2011), http://eprint.iacr.org/2011/706
5. Brands, S.: An efficient off-line electronic cash system based on the representation problem. Technical Report CS–R9323
6. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Çetin Kaya., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 292–302. Springer, Berlin, Germany, Worcester, Massachusetts, USA (Aug 12–13, 1999)
7. Dodis, Y., Pietrzak, K.: Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 21–40. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)
8. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: 49th FOCS. pp. 293–302. IEEE Computer Society Press, Philadelphia, Pennsylvania, USA (Oct 25–28, 2008)
9. Faust, S., Kiltz, E., Pietrzak, K., Rothblum, G.N.: Leakage-resilient signatures. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 343–360. Springer, Berlin, Germany, Zurich, Switzerland (Feb 9–11, 2010)
10. Faust, S., Pietrzak, K., Schipper, J.: Practical leakage-resilient symmetric cryptography. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 213–232. Springer, Berlin, Germany, Leuven, Belgium (Sep 9–12, 2012)
11. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: the computationally-bounded and noisy cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Berlin, Germany, French Riviera (May 30 – Jun 3, 2010)
12. Fouque, P.A., Tibouchi, M.: Indifferentiable hashing to Barreto-Naehrig curves. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 1–17. Springer, Berlin, Germany, Santiago, Chile (Oct 7–10, 2012)
13. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics, Vol 156 #14 pages 3113–3121 (2008)
14. Galbraith, S.D., Scott, M.: Exponentiation in pairing-friendly groups using homomorphisms. In: Pairing-Based Cryptography–Pairing 2008, pp. 211–224. Springer (2008)
15. Galindo, D., Großschädl, J., Liu, Z., Vadnala, P.K., Vivek, S.: Implementation and evaluation of a leakage-resilient ElGamal key encapsulation mechanism. Cryptology ePrint Archive, Report 2014/835 (2014), http://eprint.iacr.org/2014/835
16. Galindo, D., Vivek, S.: A practical leakage-resilient signature scheme in the generic group model. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 50–65. Springer, Berlin, Germany, Windsor, Ontario, Canada (Aug 15–16, 2012)
17. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 19–23, 2001)
18. Hazay, C., López-Alt, A., Wee, H., Wichs, D.: Leakage-resilient cryptography from minimal assumptions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 160–176. Springer, Berlin, Germany, Athens, Greece (May 26–30, 2013)
19. Kiltz, E., Pietrzak, K.: Leakage resilient ElGamal encryption. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 595–612. Springer, Berlin, Germany, Singapore (Dec 5–9, 2010)
20. Kocher, P.: Blind signature systems. U.S. Patent #4,759,063
21. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996)
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)
23. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards. Springer (2008)
24. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Berlin, Germany, Cirencester, UK (Dec 19–21, 2005)
25. Medwed, M., Standaert, F.X., Großschädl, J., Regazzoni, F.: Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 10. LNCS, vol. 6055, pp. 279–296. Springer, Berlin, Germany, Stellenbosch, South Africa (May 3–6, 2010)
26. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Berlin, Germany, Cambridge, MA, USA (Feb 19–21, 2004)
27. Miller, V.S.: The Weil pairing, and its efficient calculation. Journal of Cryptology 17(4), 235–261 (2004)

28. Nechaev, V.I.: Complexity of a determinate algorithm for the discrete logarithm. Mathematical Notes 55(2), 165–172 (1994)
29. Nguyen, P.Q., Shparlinski, I.: The insecurity of the digital signature algorithm with partially known nonces. Journal of Cryptology 15(3), 151–176 (2002)
30. Pereira, G.C.C.F., Simplício Jr., M.A., Naehrig, M., Barreto, P.S.L.M.: A family of implementation-friendly BN elliptic curves. Cryptology ePrint Archive, Report 2010/429 (2010), http://eprint.iacr.org/2010/429
31. Renauld, M., Standaert, F.X., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A formal study of power variability issues and side-channel attacks for nanoscale devices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 109–128. Springer, Berlin, Germany, Tallinn, Estonia (May 15–19, 2011)
32. Schipper, J.: Leakage-Resilient Authentication. Ph.D. thesis, Utrecht University (2010)
33. Scott, M.: Miracl–multiprecision integer and rational arithmetic C/C++ library. Shamus Software Ltd, Dublin, Ireland (2003)
34. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Berlin, Germany, Konstanz, Germany (May 11–15, 1997)
35. Tang, F., Li, H., Niu, Q., Liang, B.: Efficient leakage-resilient signature schemes in the generic bilinear group model. In: Huang, X., Zhou, J. (eds.) Information Security Practice and Experience - 10th International Conference, ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8434, pp. 418–432. Springer (2014)
36. Yacobi, Y.: A note on the bilinear Diffie-Hellman assumption. Cryptology ePrint Archive, Report 2002/113 (2002), http://eprint.iacr.org/2002/113

A Additional Figures and Tables

```
proc $\mathcal{O}_1(\xi_1, \xi_2)$:
if $\xi_1 \notin \mathcal{L}_1$ then
    $F_1 \leftarrow Guess_1(\xi_1)$
end if
if $\xi_2 \notin \mathcal{L}_1$ then
    $F_2 \leftarrow Guess_1(\xi_2)$
end if
get $F_1$ and $F_2$ from $\mathcal{L}_1$
$F_3 \leftarrow F_1 + F_2$
if $F_3 \in \mathcal{L}_1$ then
    get $\xi_3$ from $\mathcal{L}_1$
else
    $\xi_3 \leftarrow Sample_1(F_3)$
end if
Return $\xi_3$
```

```
proc $\mathcal{O}_e(\xi_1, \xi_2)$:
if $\xi_1 \notin \mathcal{L}_1$ then
    $F_1 \leftarrow Guess_1(\xi_1)$
end if
if $\xi_2 \notin \mathcal{L}_2$ then
    $F_2 \leftarrow Guess_2(\xi_2)$
end if
get $F_1$ from $\mathcal{L}_1$
get $F_2$ from $\mathcal{L}_2$
$F_3 \leftarrow F_1 \cdot F_2$
if $F_3 \in \mathcal{L}_3$ then
    get $\xi_3$ from $\mathcal{L}_3$
else
    $\xi_3 \leftarrow Sample_3(F_3)$
end if
Return $\xi_3$
```

```
proc $Sample_1(F)$:
$\xi \xleftarrow{\$} \Xi_1 \backslash \mathcal{L}_1$
add $(F, \xi)$ to $\mathcal{L}_1$
Return $\xi$

proc $Guess_1(\xi)$:
$d \xleftarrow{\$} \mathbb{Z}_p$
add $(d, \xi)$ to $\mathcal{L}_1$
Return $d$
```

Fig. 8: GGM group oracles used within the proof ($\mathcal{O}_2, \mathcal{O}_3, Sample_2, Sample_3, Guess_2, Guess_3$ are not included due to their similarity to the oracles for $\mathbb{G}_1$.)

```
Random_Sampler:
loop
    $x \xleftarrow{\$} \mathbb{F}_q^*$
    if $x \in E(\mathbb{F}_q)$ then
        $i \xleftarrow{\$} \{0, 1\}$
        $y \leftarrow -1^i \cdot \sqrt{x^2 + b}$
        return $R = (x, y)$
    end if
end loop
```

```
Try_and_Increment:
$x \xleftarrow{\$} \mathbb{F}_q^*$
loop
    if $x \in E(\mathbb{F}_q)$ then
        $i \xleftarrow{\$} \{0, 1\}$
        $y \leftarrow -1^i \cdot \sqrt{x^2 + b}$
        return $R = (x, y)$
    else
        $x \leftarrow x + 1$
    end if
end loop
```

```
Random_Scalar:
Let $P$ be a generator of $E(\mathbb{F}_q)$
$k \xleftarrow{\$} \mathbb{F}_q^*$
return $R = [k]P$
```

```
SWEncoding:
$t \xleftarrow{\$} \mathbb{F}_q^*$
$w \leftarrow \sqrt{-3} \cdot t/(1 + b + t^2)$
$x_1 \leftarrow -1 + \sqrt{-3}/2 - tw$
$x_2 \leftarrow -1 + x_1$
$x_3 \leftarrow 1 + 1/w^2$
$r_1, r_2, r_3 \xleftarrow{\$} \mathbb{F}_q^*$
$\alpha \leftarrow \chi_q(r_1^2 \cdot (x_1^3 + b))$
$\beta \leftarrow \chi_q(r_2^2 \cdot (x_2^3 + b))$
$i \leftarrow [(\alpha - 1) \cdot \beta \mod 3] + 1$
$R \leftarrow (x_i, \chi_q(r_3^2 \cdot t) \cdot \sqrt{x_i^3 + b})$
return $R \in E(\mathbb{F}_q)$
```

Fig. 9: The algorithms for the four point samplers considered

B Comparison of our Scheme

In this section we focus on some practical considerations: how efficient is it in comparison to other leakage resilient MAC constructions, and what would a practical implementation need to guarantee to meet our leakage bound/assumptions?

Before giving a comparision we need to make some choices for parameters of the various schemes. In case of schemes which have as underlying primitive a pseudo random function (PRF), we chose to instantiate this PRF with AES-128. This is motivated by the fact that this reflects the current state of the art. For our own scheme, and a somewhat comparable signature scheme, we use as instantiation of

Table 3: Performance results of underlying operations.

Device	Cortex-M4		Intel i7	
Operation	Time (ms)		Time (μs)	
	128-bits	192-bits	128-bits	192-bits
Squaring in $\mathbb{G}_1$	< 1	< 1	< 1	< 1
Squaring in $\mathbb{G}_2$	< 1	< 1	~ 1	~ 2
Squaring in $\mathbb{G}_3$	~ 1	~ 1	~ 11	~ 12
Pairing	1864	26327	7151	60811
Hashing (21 Bytes)	280	3987	699	7622

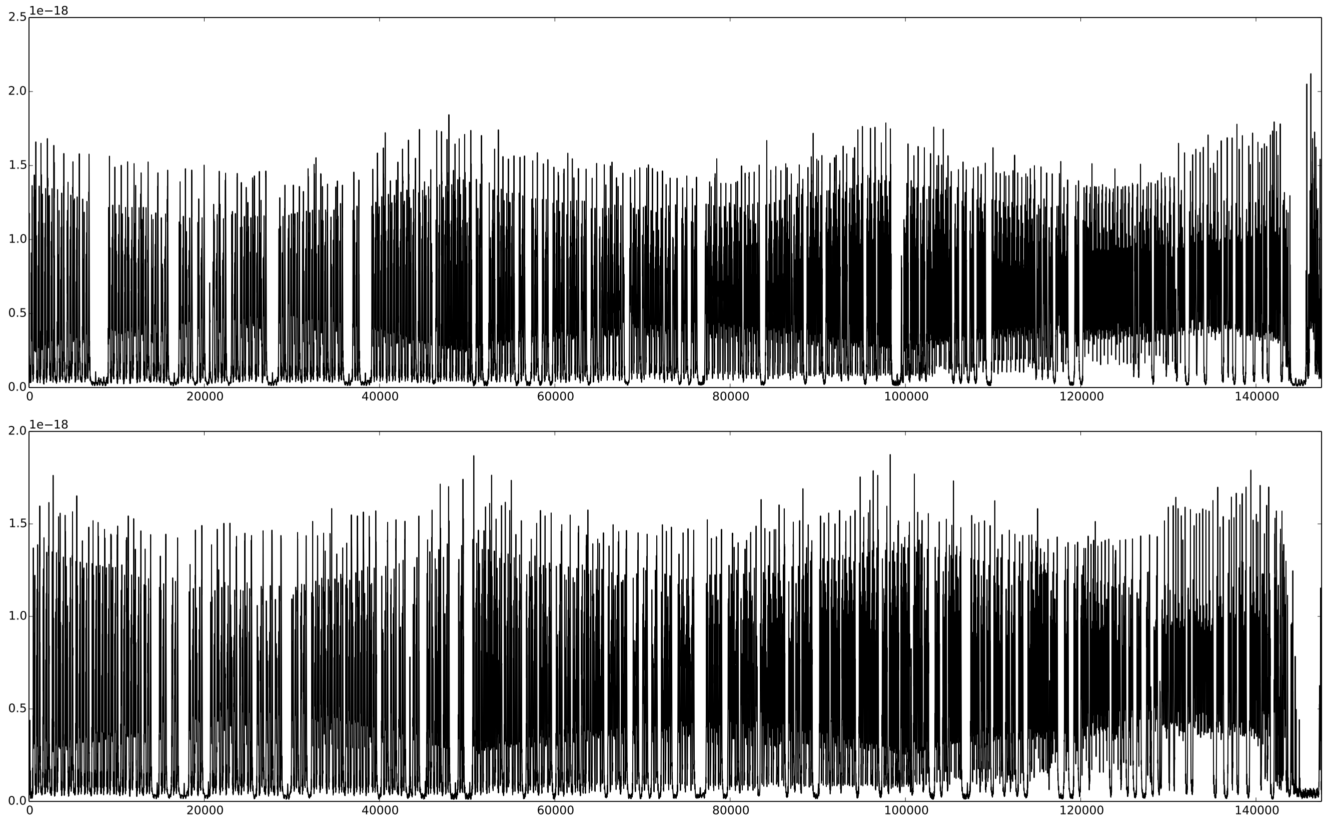

Fig. 10: Two example trace of the Jabobi symbol being calculated.

the bilinear map a pairing defined over a suitable pairing friendly elliptic curve. In this case we choose as group size parameter 2^{160}, again with the motivation to reflect a state-of-the art security bound (i.e. 2^{80} is regarded as minimum for bound which is implied by a group of double this size).

B.1 Security considerations

The leakage bound we have for our scheme is that we can tolerate up to approx. 50 bits of leakage assuming a group size of about 2^{160} per invocation of the scheme. For a practical implementation it is important to acknowledge that this will not include the initial sharing out of the key. Consequently in a strict sense this would need to be done in a secure environment.

When considering what '50 bits of leakage' means given our constructions, it helps to think about the tagging and verification algorithms in a concrete instantiation: i.e. we are working with pairings that are defined over some pairing friendly curve. Consequently, $S_i^{\text{◓}}$, $S_i^{\text{◒}}$, r_i and ${r_i}^{-1}$ are elliptic curve points, $S_i^{\text{◓}} \cdot r_i$ and $S_i^{\text{◒}} \cdot {r_i}^{-1}$ are elliptic curve point additions, and e is a pairing. Now that we have a concrete instantiation of our scheme, we can argue more concretely about implementation options. In order to provide some resistance against simple side channel attacks such as SPA we would hence essentially need to blind the EC points which correspond to secrets. A recent contribution [4] gives a good overview on various side channel attacks on pairings from which one can again conclude that also the pairing operation should best be implemented on blinded EC points.

The obvious question arising now might be what we have gained in practice, as we yet again need to apply the basic techniques that would prevent attacks such as SPA and DPA? The answer is for any non leakage resilient version, even partial leakage on the random values will probably allow lattice based attacks such as [29]. The leakage resilient scheme however guarantees that in the presence of partial leakage no such attack can succeed.

B.2 Comparison with other leakage-resilient schemes

In our comparison we essentially look at the number of elliptic curve or AES operations that tagging and verification require. We also report on the tolerated leakage, the key size and the tag size. Table 4 provides an overview and the following text explains and discusses the provided numbers. As can be seen from this table, our scheme is highly competitive when compared with other provably secure schemes.

The only leakage resilient MAC scheme in the literature that isn't built on top of another leakage resilient scheme is by Hazay *et al.* [18] to the best of our knowledge. The advantage of their work is that it only relies on very minimal assumptions (the existence of one-way functions). However the leakage bound is a total bound, *i.e.* it holds regardless of the number of times the tag and verify algorithms are called. Assuming that we instantiate the PRF required for this scheme with AES-128 (*i.e.* setting $\lambda = s = 128$ to use the notation in the paper), and using the equations they give in Theorem 5.6, it turns out that AES will be called 512 times per tag and verify query. While this already makes the scheme computationally expensive, the larger problem is the overall key size: the key must be of size approximately 2^{18} bits, which is impractical for many applications. Even worse, under these parameters the MAC can leak 512 bits over the lifetime of the system (i.e. not per MAC invocation). This also means that with these choices AES and the elliptic curve schemes will give approximately the same level of security.

Due to the small number of leakage resilient MACs, we also provide a comparison against PRFs and signatures because although there is not an immediate strategy to convert them into leakage resilient MACs available, there is potential that one might use PRFs or signatures to instantiate a MAC with some leakage resilient properties.

The PRF by Dodis and Pietrzak [7] requires that the leakage functions are fixed prior to the attack and are not adaptively chosen. They define the PRF $\Gamma^F : \Sigma^{3k+n} \times \Sigma^m \to \Sigma^{4k+2n}$ created from a wPRF $F : \Sigma^k \times \Sigma^n \to \Sigma^{4k+2n}$. If we instantiate F with AES-128 with something like $F(x) = Enc_K(x||000)||Enc_K(x||001)||Enc_K(x||010)||Enc_K(x||011)||Enc_K(x||100)||Enc_K(x||101)$ and then take the desired number of output bits, this gives us $k = m = 128$, $n = 125$ and $\Gamma^F : \Sigma^{509} \times \Sigma^{128} \to \Sigma^{768}$. As stated in the paper each time the PRF is called, the function F is called $m + 1$ times and thus AES will be called 774 times. Hence even if this can be converted into a MAC reasonably inexpensively, the PRF itself is very expensive.

Schipper's construction [32] requires the use of a EUF-CMA MAC and a leakage resilient PRF. Hence, the timings will be very similar to the numbers from [7] and thus we do not include it explicitly in the table.

The PRF by Faust *et al.* [10] requires that neither the leakage or the PRF inputs are queried adaptively but are fixed prior to the start of the game. They construct a scheme $\Gamma^{F,m} : \{0,1\}^{k+(m+1)l} \times \{0,1\}^m \to \{0,1\}^n$ which uses the wPRF $F : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^{2k}$ and $m + 1$ public random values of length l. If we again instantiated the wPRF with AES-128 to get $F(x) = Enc_K(x||0)||Enc_K(x||1)$ meaning we get $m = k = 128$, $l = 127$,$n = 256$ and $\Gamma^{F,128} : \{0,1\}^{16511} \times \{0,1\}^{128} \to \{0,1\}^{256}$. This will call AES 258 times per invocation, which while an improvement still seems prohibitive for a practical implementation.

The signature scheme by Faust *et al.* [9] uses $2l$ exponentiations, $4(l-1)$ multiplications, $l-1$ additions and 2 hash function calls in the signing algorithm and tl exponentiations, tl multiplications and t hash function calls in the verification algorithm, where l is related to the underlying l-representation problem [5] (assumed to be hard) and t is the depth of the signature chain. The downside of this scheme is that even if $l = 2$, while signing is efficient, verification takes longer depending how deep the signature chain is. This could mean that verification quickly becomes too expensive for an embedded device to perform.

The signature scheme by Galindo and Vivek [16] is the only one comes close to our construction in terms of performance, which is unsurprising given that it is also based on the same key update mechanism. For signing, the algorithm uses 2 Elliptic Curve scalar multiplications, 5 Elliptic Curve additions and also generates a random curve point (and its' inverse), while verification uses 2 Elliptic Curve point additions, 1 Elliptic Curve scalar multiplication and 2 pairings. Since a pairing is currently only slightly more expensive than an exponentiation, our tagging algorithm will be almost equivalent in timing to their signing algorithm. Their verification is faster than ours, however, their keys are larger. Furthermore, while there is no hash function explicit in their scheme, it is assumed that the message comes from $\mathbb{Z}_p$ and thus in practice to sign arbitrary messages, the message would have to be hashed onto $\mathbb{Z}_p$.

Table 4: A comparison of possible EUF-CMA-LT schemes

Scheme	Leakage	Key Size	Tag Size	Tag Time	Verification Time
Our scheme	$\frac{1}{2}(\log p - 2\log q_u - n)$ p group size q queries n security bound (approx 50 bits)	2 EC points (approx 320 bits)	1 EC point 1 random string (approx 228 bits)	1 hash 1 random point (and inverse) generated 3 EC additions 2 pairings	1 hash 1 random point (and inverse) generated 3 EC additions 2 pairings
HLWW:[18]	512 bits total	approx 2^{18} bits	approx 2^{19} bits	512 AES calls	512 AES calls
DP:[7]	$\frac{\log(\epsilon^{-1})}{6}$ for $\epsilon = \mathbf{Adv}_F^{wprf}(A)$ (approx 22 bits)	509 bits	762 bits	774 AES calls	774 AES calls
FPS:[10]	$\frac{\log(\epsilon^{-1})}{4}$ for $\epsilon = \mathbf{Adv}_F^{wprf}(A)$ (approx 32 bits)	128 bits 16,383 public bits	256 bits	258 AES calls	258 AES calls
FKPR:[9]	2kl$(\frac{1}{2} - \frac{1}{2l} - \delta)$ (approx 40 bits)	Varies	Varies	$2l$ exponentiations $4(l-1)$ multiplications $l-1$ additions 2 hashes	Varies
GV:[16]	$<< \frac{\log p}{2}$ p is group size (approx 60 bits)	sk: 2 EC points pk: 3 EC points (approx 800 bits)	2 EC points (approx 320 bits)	1 random point (and inverse) generated 2 scalar multiplications 5 EC additions 1 hash	1 scalar multiplication 2 EC additions 2 pairings 1 hash