Evaluation Report on the **ACE Encrypt Cryptosystem**

1 Introduction

This document is an evaluation of the **ACE Encrypt Cryptosystem**. Our work is based on the analysis of document [27], which provides both the specification and self-evaluation of the scheme, as well as on the research papers [7, 25, 26], where additional security arguments can be found. The present report is organized as follows: firstly, we briefly review the cryptosytem; next we discuss the security level of the cryptographic primitive which underlies the scheme and analyze its relation to the difficulty of the discrete logarithm problem; finally, we evaluate the security level of the scheme itself in the light of strong security notions such as semantic security and security against adaptive chosen-ciphertext attacks. This is as requested by IPA.

2 Brief description of the scheme

2.1 Specification review

ACE Encrypt is based on the hardness of the discrete logarithm problem. It is a variant of an hybrid encryption scheme derived from the Cramer Shoup cryptosystem, which has appeared in [26]. The public key encryption scheme of Cramer and Shoup itself (see [7]), is an extension of the El Gamal (see [10]) cryptosystem. To make things more precise, we need some notation from [27]: P is a prime integer whose size in bits is m, $1024 \leq m \leq 16,384$ and q is an 256-bit prime number which is a divisor of $P-1$. Let G be the subgroup of $\mathbb{Z}_P^\star$ consisting of elements of multiplicative order q. The public key of ACE Encrypt includes a pair of integers of the form P, q, and six elements of the corresponding group G, denoted by g_1, g_2, c, d, h_1, h_2. There are two additional items in the public key, consisting of bit strings k_1, k_2, of appropriate length that are used as keys for various hash functions. We will bring more precision on these elements at a later stage of our report.

Before going further, we introduce a more formal framework, that will be useful when we later perform the security analysis. A public-key encryption scheme on a message space $\mathcal{M}$ consists of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:

- the key generation algorithm $\mathcal{K}(1^m)$ outputs a random pair of secret-public keys $(\mathsf{sk}, \mathsf{pk})$, relatively to a security parameter m
- the encryption algorithm $\mathcal{E}_{\mathsf{pk}}(M; r, s)$ outputs a ciphertext C corresponding to the plaintext $M \in \mathcal{M}$ using random coins r, s
- the decryption algorithm $\mathcal{D}_{\mathsf{sk}}(C)$ outputs the plaintext m associated to the ciphertext C.

Thus, the key generation algorithm $\mathcal{K}(1^k)$ of the ACE Cryptosystem produces, on input m, a public key pk consisting of P, q and the six elements g_1, g_2, c, d, h_1, h_2, as well as the additional bit strings k_1, k_2. The secret key sk consists of the respective logarithms w, x, y, z_1, z_2 of g_2, c, d, h_1, h_2 in base g_1. We refer to [27] for a precise definition of the length of k_1, k_2 in terms of the security parameter. We now turn to encryption and decryption.

Encryption $\mathcal{E}_{\mathsf{pk}}(M; r, s)$ uses several hash functions and produces a preamble and a cryptogram. The preamble consists of a tuple (s, u_1, u_2, v). Its first component s is a random 128-bit string. The next two elements u_1, u_2 are computed from a randomly chosen r, $0 \leq r \leq q-1$ by $u_1 = g_1^r$, $u_2 = g_2^r$, where operations are performed in group G. Finally, v is obtained by hashing the triple (s, u_1, u_2), using a hash function keyed by k_1. Denoting this function by H_1, we have $\alpha = H_1(s, u_1, u_2)$ and, finally, $v = (cd^\alpha)^r$.

At this point, we wish to note that document [27] has an extremely precise notational apparatus. For example, it would use

$$\alpha = UOWhash'(k_1, L_{\mathbf{B}}(P), s, u_1, u_2)$$

so as to emphasis the functional role of H_1 (a universal one-way hash funstion) and its dependency on a hash key k_1 as well as upon the size in bytes, $L_{\mathbf{B}}(P)$, of parameter P. This is perfect, at specification level, to avoid implementation errors, but cumbersome for performing a high level security analysis. This is why we use simplified notations. Similarly, we ignore type conversion and consider s as a 128-bit string, whereas document [27] treats it as four 32-bit words.

With respect to the Cramer-Shoup system proposed at Crypto'98 (see[7]), the preamble includes the additional random element s, which is later used, as an "initialization vector" in generating the cryptogram. A 128-bit symmetric key k is derived from s, u_1, $\tilde{h}_1$, $\tilde{h}_2$, where $\tilde{h}_1 = h_1^r$, $\tilde{h}_2 = h_2^r$ and r is the random integer used to create the preamble. This is by means of a hash function keyed by the second hash key k_2, which we denote as H_2. Thus, we write $k = H_2(s, u_1, \tilde{h}_1, \tilde{h}_2)$. Note that k is uniquely defined from the preamble. However, recovering k from (s, u_1, u_2, v) appears closely connected to the discrete logarithm problem in group G. The requirement on H_2 is that it is *universal* in terms of the inputs $\tilde{h}_1, \tilde{h}_2$: for every pair x, y of such inputs, $x \neq y$, the probability (over the key k_2) that $H_2(x) = H_2(y)$ is $1/2^\ell$, where $\ell = 256$ is the size of the outputs of H_2.

Decryption $\mathcal{D}_{\mathsf{sk}}(C)$ is precisely based on recovering k: granted these secret values, it is possible to check the correctness of the preamble through the following equalities

$$u_2 = (u_1)^w$$
$$v = (u_1)^{x+y\alpha}$$

Il is also possible to derive $\tilde{h}_1$, $\tilde{h}_2$ by

$$\tilde{h}_1 = u_1^{z_1}$$
$$\tilde{h}_2 = u_1^{z_2}$$

If all logarithms are unknown, the above computations are not possible. In order to grasp the relationship between decryption and the discrete logarithm, it is interesting to consider the situation where part of the secret key is disclosed, while the value of w remains concealed. We thus introduce the following definition:

Definition. An *incomplete key* consists of elements x_1, x_2, y_1, y_2, z_{11}, z_{12}, z_{21}, z_{22} such that

$$c = g_1^{x_1} g_2^{x_2}$$
$$d = g_1^{y_1} g_2^{y_2}$$
$$h_1 = g_1^{z_{11}} g_2^{z_{12}}$$
$$h_2 = g_2^{z_{21}} g_2^{z_{22}}$$

Note that it is perfectly possible to generate elements g_1, g_2, c, d, h_1, h_2 of a public key without disclosing the discrete logarithm of g_2 in base g_1 by choosing x_1, x_2, y_1, y_2, z_{11}, z_{12}, z_{21}, z_{22} at random in the interval $\{0, \cdots, q-1\}$ and producing the corresponding incomplete key. Note also that, with an incomplete secret key, it is still possible to obtain $\tilde{h}_1$, $\tilde{h}_2$ as

$$\tilde{h}_1 = u_1^{z_{11}}.u_2^{z_{12}}$$
$$\tilde{h}_2 = u_1^{z_{21}}.u_2^{z_{22}}$$

and to check the value of v by

$$v = (g_1)^{x_1+\alpha y_1} (g_2)^{x_2+\alpha y_2}$$

However, it is not possible to perform the crucial check $u_2 = (u_1)^w$, since w is missing. This already has interesting consequences in terms of security against a passive adversary.

Theorem 1 *Based on the decisional Diffie-Hellman hypothesis (DDH), it is hard to distinguish the distribution*

$$(g_1, g_2, u_1, u_2, v, h_1, h_2, \tilde{h}_1, \tilde{h}_2)$$

generated by the cryptosystem, from the analogous distribution with $\tilde{h}_1$, $\tilde{h}_2$ replaced by random elements of G. More accurately, if there is an adversary $\mathcal{A}$ that distinguishes the above distributions within time bound t, with advantage ε, then there exists a machine $\mathcal{B}$ that solves the decisional Diffie-Hellman problem with advantage $\varepsilon/2$ within time bound $t + \tau$, where τ accounts for a few extra modular exponentiations and is bounded by $\mathcal{O}(m^2 \log q)$.

In other words, the trapdoor information $\tilde{h}_1, \tilde{h}_2$ is semantically secure in the sense of the seminal paper [13]. In the above, the advantage in distinguishing two distributions is the absolute value of the difference of the probabilities that the algorithm outputs 1, with inputs taken from each.

Proof. Let $\mathcal{A}$ be an adversary that distinguishes the two distributions defined in the theorem. We show how to attack the DDH by distinguishing the distributions $\mathbf{D}$ and $\mathbf{R}$, where

$$\mathbf{R} = \{(g_1, g_2, u_1, u_2)\}$$

with all four elements taken at random in G and

$$\mathbf{D} = \{(g_1, g_2, u_1, u_2)\}$$

with $\log_{g_1}(u_1) = \log_{g_2}(u_2)$. We generate an incomplete secret key x_1, x_2, y_1, y_2, z_{11}, z_{12}, z_{21}, z_{22}. Next, we compute the public key from g_1, g_2 and the incomplete secret key and form

$$(g_1, g_2, u_1, u_2, v, h_1, h_2)$$

as explained above, computing v as

$$v = (g_1)^{x_1 + \alpha y_1} (g_2)^{x_2 + \alpha y_2}$$

Then, we flip a random bit b; if $b = 0$, we compute

$$\gamma_1 = u_1^{z_{11}}.u_2^{z_{12}}$$

$$\gamma_2 = u_1^{z_{21}}.u_2^{z_{22}}$$

whereas, if $b = 1$ we choose them randomly. Finally, we handle the t-uple

$$(g_1, g_2, u_1, u_2, v, h_1, h_2, \gamma_1, \gamma_2)$$

to $\mathcal{A}$. When the input (g_1, g_2, u_1, u_2) is from $\mathbf{D}$, the output b' equals b with probability $\frac{1 \pm \varepsilon}{2}$. On the other hand, when (g_1, g_2, u_1, u_2) is from $\mathbf{R}$, since z_{11}, z_{12}, z_{21}, z_{22} are chosen randomly and independently, the same distribution is handled to $\mathcal{A}$, regardless of the value of b. Thus, we have obtained a distinguisher between $\mathbf{D}$ and $\mathbf{R}$, with advantage $\varepsilon/2$. Thus, using notations similar to those of [27], the advantage of any algorithm $\mathcal{A}$ that runs in time t is bounded by $2\mathtt{AdvDDH}(O(t))$, where $O(t) = t + \tau$ accounts for the few extra operations needed to compute the data to be handled to $\mathcal{A}$.

We now turn to the symmetric encryption step. The cryptogram e comes from the cleartext M. Encryption is performed by means of a stream cipher based on the IBM AES candidate MARS. MARS is used in sum/counter mode: let $f(k,x)$ denote the 128-bit output of MARS when encrypting a 128-bit input x under a 256-bit key k. A bit stream is produced step by step: at the i-th step, 128 bits are output as:

$$f(k, s+2i) \oplus f(k, s+2i+1)$$

where addition is modulo 2^{128} and $\oplus$ is bitwise exclusive or. The symmetric encryption part has an extra feature. Namely, it includes a MAC for every 1024-byte block. The bit stream output by MARS is thus used:

1. to generate a hash key $\overline{k}$
2. to generate enough key material to mask the successive blocks and their respective MACs

The MAC is computed from each ciphertext block enc, by means of a third hash function H_3, keyed by $\overline{k}$, as $H_3(f, enc)$, where f is a flag whose value is 1 for the last block and 0 otherwise. Hash function H_3 itself is the composition of two keyed hash function H_{31} and H_{32}, where H_{31} is essentially similar to H_1 and H_{32} is simply a universal hash function in the sense of Carter and Wegman (see[8] and also [19]).

It is expected that both H_1 and H_{31} are one-way universal (granted the intractability hypotheses). This means that it is hard for an adversary $\mathcal{A}$ to win the following game:

1. $\mathcal{A}$ chooses a message x
2. $\mathcal{A}$ receives a random key K for H
3. $\mathcal{A}$ outputs a message y and wins if it has found a collision, i.e. $y \neq x$ and $H(K,y) = H(K,x)$.

2.2 Comments on the specification

Document [27] is extremely well written and should actually ensure interoperability between different implementations, as claimed. There are very few ambiguities. Our only minor criticism is that some names are used both for "global" and "local" variables, which might be puzzling for implementors. For example m is used in section **4.1** to denote the bit length of P, whereas it appears in section **4.4** as the length in bytes of the blocks for symmetric encryption.

In terms of security analysis, the picture is slightly less convincing. By this, we do not mean that the security arguments that are offered are wrong. We will discuss them in more detail further on in the present report and, as will be seen, they are mathematically sound. However, they are certainly elaborate and, presumably, can be understood by a fraction only of the research community in cryptography. As a result, the specification lacks flexibility and can only be taken as a whole. Simple questions on the design rationale cannot be answered without going deeply into the security analysis. We give a few examples:

1. Why does document [27] introduce two values h_1, h_2 whereas the research paper [26] on which it builds has only one such value h?
2. What happens if one derives the symmetric encryption key k by applying $SHA1$ to the data s, u_1, $\tilde{h_1}$, $\tilde{h_2}$ instead of using the function proposed in the specification?
3. What happens if one substitutes another symmetric encryption scheme to the one specifically described in the specification?

These questions may seem outside the requested security analysis. Nevertheless, we believe that they are important: security standards have to cope with higher level specifications and with practical constraints. Thus, the analysis should preferably include indications of how the security level degrades if "minor" changes are performed.

3 Security level of the cryptographic primitive

In this section, we investigate the security of the underlying cryptographic primitive, both in terms of complexity-theoretic reductions and with respect to the recommended parameters.

3.1 Complexity-theoretic arguments

Document [27], measures the security of the scheme in terms of

1. the hardness of the decisional Diffie-Hellman hypothesis
2. the second preimage resistance of the core compression function of the hash function $SHA1$
3. the pseudo-randomness of the AES candidate MARS used in sum/counter mode

We comment on these basic primitives.

3.1.1 The decisional Diffie-Hellman hypothesis

We keep the notations of section 2.1. Recall that the decisional Diffie-Hellman hypothesis over a group G asserts that it is hard to distinguish the distributions $\mathbf{D}$ and $\mathbf{R}$, where

$$\mathbf{R} = \{(g_1, g_2, u_1, u_2)\}$$

with all four elements taken at random in G and

$$\mathbf{D} = \{(g_1, g_2, u_1, u_2)\}$$

with $\log_{g_1}(u_1) = \log_{g_2}(u_2)$. A quantitative version measures the maximum advantage $\texttt{AdvDDH}(t)$ of a statistical test T that runs in time t. This means the maximum of the difference of the respective probabilities that T outputs 1, when probabilities are taken over $\mathbf{D}$ or $\mathbf{R}$.

As explained in [27], there is a standard self-reducibility argument: by randomization, it is possible to transform an arbitrary tuple (g_1, g_2, u_1, u_2) such that $g_1 \neq g_2$ into a random equivalent one, i.e. the output is in $\mathbf{D}$ (resp. $\mathbf{R}$), iff and only if the input is (see [27], section **2.4**.) Thus, if $\texttt{AdvDDH}(t)$ is significant, one can use a distinguisher to decide, with probability close to one, whether a tuple is in $\mathbf{D}$. This involves performing repeated tests with the distinguisher, and deciding whether the number of one outputs has a bias towards $\mathbf{D}$ or $\mathbf{R}$. Based on the law of large numbers, a decision with small constant error probability requires running $O(\texttt{AdvDDH}^{-2})$ tests. One can decrease the error probability drastically by repeating the above computations an odd number of times and deciding based on the median of the averages. In [21], the authors claim that one can reach error probability 2^{-n} by repeating the test $O(p(n)).\texttt{AdvDDH}^{-1}$, where p is a polynomial, but the proof is missing. In any case, the loss in the reduction is huge. Thus, despite its elegance, the self-reducibility argument is a bit misleading in terms of exact security.

Related to the above is the computational Diffie-Hellman assumption (CDH) and the discrete logarithm assumption. The former states that it is hard to compute g^{xy} from g, g^x and g^y, while the latter states that it is hard to compute x from g and g^x. It is obvious that DDH is a stronger assumption than CDH, which in turn, is stronger than the discrete logarithm assumption. However, no other relation is known and the only way to solve the hard problems underlying DDH or CDH is to compute discrete logarithms. There are some indications, in other settings, that DDH might be easy, in some cases, while the CDH remains difficult (see [12] and [11]). However, the references just quoted do not seem to be relevant to the present context.

3.1.2 The second preimage resistance of SHA1

Here, we note that second preimage resistance is weaker than collision resistance. Furthermore, contrary to finding collisions, there is no known algorithm that computes a second preimage of a hash function in time $O(\sqrt{S})$, where S is the size ot its range. Thus, SHA1 might survive in this setting, even once it has become obsolete as a collision-free hash function and replaced –say– by the forthcoming SHA-256.

3.1.3 Using MARS as a pseudo-random generator

We will not discuss the security of MARS. During the AES process, no significant weakness of MARS was found.

3.2 Size of the parameters

As observed above, the only method known to attack the decisional Diffie-Hellman problem is to solve the underlying discrete logarithm problem (DLP). In order to estimate whether the suggested parameters of the scheme offer a wide security margin, it is useful to review the performances of the various algorithms known for the DLP. Considering that the scheme uses a group G which is a subgroup of some $\mathbb{Z}_P^\star$, we will distinguish between exponential algorithms, whose running time depends on the size of the group and subexponential algorithms, where the estimate is in terms of the size of P

3.2.1 Exponential algorithms

There is an algorithm due to Pohlig and Hellman (see [22]), which reduces the determination of the discrete logarithm modulo a prime P to the analogous problem modulo each of the prime factors of $P-1$. Such an algorithm is efficient if $P-1$ is a product of small primes but irrelevant to the present context, since q is a large prime.

The best algorithm known to date for solving the DLP in any given group G is the Pollard ρ-method from [23] which takes computing time equivalent to about $\sqrt{\pi n/2}$ group operations. In 1993, van Oorschot and Wiener in [28], showed how the Pollard ρ-method can be parallelized so that, if t processors are used, then the expected number of steps by each processor before a discrete logarithm is obtained is $\simeq \frac{\sqrt{\pi n/2}}{t}$. In order to compute the discrete logarithm of y in base g, each processor computes a kind of random walk within elements of the form $g^a y^b$, selecting x_{i+1} as yx_i or x_i^2 or else gx_i, according to a deterministic but randomly looking choice (say based on a hash value). "Distinguished" points x_i are stored together with their representations $x_i = g^{a_i} y^{b_i}$ in a list that is common to all processors. When a collision occurs in the list, the requested discrete logarithm becomes known. There is no record involving discrete logarithms in subgroups of $\mathbb{Z}_P^\star$. However, one can estimate what such a record would be, by recalling the current record for computations in the group of points of an elliptic curve. In april 2000, the solution to the ECC2K-108 challenge from Certicom led to the computation of a discrete logarithm in a group with 2^{109} elements. This is one of the largest effort ever devoted to a public-key cryptography challenge. The amount of work required to solve the ECC2K-108 challenge was about 50 times that required to solve the 512-bit RSA cryptosystem (see [4]) and was thus close to 400000 mips-years. Because, the standard arithmetical operations execute faster than elliptic curve additions, an equivalent effort in the area of subgroups of $\mathbb{Z}_P^\star$ would presumably reach a few bits more. Referring to [20], we find that it would mean an extra 4 to 5 bits. Based on [20], there is no indication that discrete logarithms of 256 bits, as suggested by the curent scheme, can be computed in the next 50 years.

3.2.2 Subexponential algorithms

Index calculus method. The index calculus method has two steps:

1. One fixes a subset of $\mathbb{Z}_P^\star$, called the factor base, and tries to write elements, whose discrete logarithm is known, as product of elements of the factor base. This produces linear relations between the elements of the factor base. When enough elements have been found, the logarithms of the factor base are obtained by linear inversion modulo $P-1$.

2. To compute the logarithm of y, one tries enough $g^a y$ until the result factors over the factor base.

This method has been extended in [6], working with an imaginary quadratic number field. The extension has been termed *Gaussian Integer Method.* Its time complexity is $L(P)^{1/2+o(1)}$ with

$$L(x) = \exp(\sqrt{\ln x \ln\ln(x)})$$

It has been used until 1998 to establish records:

- 85 digits in 1996 (see [29])
- 90 digits in 1998 (see [17])

Number field sieve. The Gaussian Integer Method has been generalized to arbitrary fields in [14, 24]. The new method NFS uses two polynomials with a common root m modulo n. These polynomials should have as many smooth values as possible. It has four steps:

1. finding “good” polynomials
2. sieving
3. inverting the linear system
4. computing specific logarithms

The time complexity of NFS is

$$O(e^{(\ln n)^{1/3}(\ln\ln n)^{2/3}(C+o(1))})$$

for a small constant C (about $(64/9)^{1/3} \simeq 1.923$). This is similar to the NFS factoring method, although the records are tens of digits behind. The current record is 100 digits (see [18]). It used the equivalent of one year of computing time of a 450 MHz Pentium.

Practical experience with the NFS method for the discrete logarithm appears more limited than what it is for factoring. In [20], the authors state that

> It is generally accepted that, for any b in the current range of interest, factoring b-bit integers takes about the same amount of time as computing the discrete logarithm in $b-x$-bit field, where x is a small constant around 20.

In [4], the authors derive the following formula

$$Y = 13.24D^{1/3} + 1928.6$$

for predicting the calendar year for factoring D-digit number by NFS. Together with the above estimate, this yields

$$Y = 13.24(D+6)^{1/3} + 1928.6$$

for predicting the calendar year for the discrete logarithm. Applying the formula with $D = 309$, i.e. for a 1024 bit modulus, produces $Y = 2019$. Similarly, one gets $Y = 2042$ for 2048-bit integers and $Y = 2070$ for 4096-bit. Since the scheme under review allows moduli up to $16,384$ bits, there is a really wide margin of security.

3.2.3 Conclusion

Based on current estimates, it appears that the proposed parameters for ACE-Encrypt remain secure for the foreseeable future, at least for the next 50 years. Progress in subexponential algorithm such as NFS is the main threat to the cryptosystem. Accordingly, the size of the modulus will have to grow in time within the proposed range, taking this progress into account.

4 Security Analysis

Document [27] includes a nicely written security analysis. However, it occasionnally refers for details to the arguments proposed in connection with earlier versions of the cryptosystem (in [7, 26]). As observed in section 2.2, there are differences between the schemes of [7, 26] and the current version. Since the arguments are a bit subtle, we have found necessary to review the security analysis.

4.1 Formal framework

An asymmetric encryption scheme is *semantically secure* if no polynomial-time attacker can learn any bit of information about the plaintext from the ciphertext, except its length. More formally, an asymmetric encryption scheme is (t, ε)-IND where IND stand for *indistinguishable*, if for any adversary $\mathcal{A} = (A_1, A_2)$ with running time bounded by t, the advantage

$$\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) = \Pr_{\substack{b \overset{R}{\leftarrow} \{0,1\} \\ r,s \overset{R}{\leftarrow} \Omega}} \left[\begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{K}(1^m), (M_0, M_1, st) \leftarrow A_1(\mathsf{pk}) \\ c \leftarrow \mathcal{E}_{\mathsf{pk}}(M_b; r, s) : A_2(c, st) \overset{?}{=} b \end{array} \right] - 1/2$$

is $< \varepsilon$, where the probability space includes the internal random coins of the adversary, and M_0, M_1 are two equal length plaintexts chosen by the adversary in the message-space $\mathcal{M}$.

Another security notion has been defined in the literature, called *non-malleability* [9]. Informally is states that it is impossible to derive, from a given ciphertext, a new ciphertext such that the plaintexts are meaningfully related. We won't dicuss this notion any further since it has been proven equivalent to semantic security in an extended attack model.

The above definition of semantic security covers passive adversaries. It is a *chosen–plaintext* or CPA attack since the attacker can only encrypt plaintext. In the extended model, the *adaptive chosen–ciphertext* or CCA attack, the adversary is given access to a decryption oracle and can ask the oracle to decrypt any ciphertext, with the only restriction that it should be different from the challenge ciphertext. It has been proven in [1] that, under CCA, semantic security and non-malleability are equivalent. This is the strongest security notion currently considered.

4.2 Security of key encapsulation

We turn to the security analysis. We want to prove that the ACE scheme is IND-CCA in the standard model of computation based on the assumption that DDH is hard and on precise statements related to the security of $SHA1$ and $MARS$. Following [26], we want to first focus on the *key encapsulation*, a term which we use with a slightly different meaning from [27]. This requires extending theorem 1 to the context of CCA–adversaries. Here the attacker $\mathcal{A}$ receives an input taken from two distributions as follows

1. either, it is taken from the distribution

$$\mathcal{D}_0 = (s, u_1, u_2, v, \tilde{h_1}, \tilde{h_2})$$

generated by the cryptosystem

2. or else from the analogous distribution $\mathcal{D}_1$ where $\tilde{h}_1$, $\tilde{h}_2$ are replaced by random elements γ_1, γ_2 of G.

The attacker $\mathcal{A}$ is allowed to query a decryption oracle by submitting the preamble of a ciphertext and receiving the corresponding pair $(\tilde{h}_1, \tilde{h}_2)$ as the answer. We establish the following exact security result where the advantage of $\mathcal{A}$ is the difference of the probabilities that $\mathcal{A}$ outputs 1.

Theorem 2 *Let $\mathcal{A}$ be a CCA–adversary $\mathcal{A}$ as above with running time $\leq t$, with advantage ε, making κ queries to the decryption oracle. There exist adversaries $\mathcal{B}_{DDH}$, $\mathcal{B}_H$, operating within time bound t' and such that $\mathcal{B}_{DDH}$ is attacking the DDH with advantage ε_{DDH} and $\mathcal{B}_H$ is playing the OWU game for H_1 with success probability ε_H, where*

$$\begin{aligned} \varepsilon_{DDH} + \varepsilon_H &\geq \frac{\varepsilon}{2} - \frac{2\kappa + 1}{q} \\ t' &\leq t + \tau \end{aligned}$$

and τ accounts for a few extra modular exponentiations and is bounded by $\mathcal{O}\left(m^2 \log q\right)$.

Proof: We use essentially the same adversary $\mathcal{B}$ as in the proof of theorem 1.

1. $\mathcal{B}$ receives its input (g_1, g_2, u'_1, u'_2); (we use dashed variables as in [27])
2. $\mathcal{B}$ generates an incomplete secret key x_1, x_2, y_1, y_2, z_{11}, z_{12}, z_{21}, z_{22}
3. $\mathcal{B}$ generates a random s', a random key k_1, and a random bit b and forms

$$(s', u'_1, u'_2, v', \gamma_1, \gamma_2)$$

where v' is computed by means of the relations

$$v' = (g_1)^{x_1 + \alpha' y_1} (g_2)^{x_2 + \alpha' y_2}$$

with $\alpha' = H_1(s', u'_1, u'_2)$ and where the definition of γ_1, γ_2 depends on the value of the random coin b. If $b = 1$, γ_1, γ_2 are chosen at random in G, whereas, if $b = 0$, one sets:

$$\gamma_1 = {u'_1}^{z_{11}} \cdot {u'_2}^{z_{12}}$$
$$\gamma_2 = {u'_1}^{z_{21}} \cdot {u'_2}^{z_{22}}$$

4. finally, $\mathcal{B}$ and runs $\mathcal{A}$ on the tuple obtained at step 3.

We have to explain how $\mathcal{B}$ simulates the decryption oracle: from the preamble of the queried ciphertext, (s, u_1, u_2, v), using the incomplete secret key, it is still possible to obtain $\tilde{h}_1$, $\tilde{h}_2$ as

$$\tilde{h}_1 = u_1^{z_{11}}.u_2^{z_{12}}$$
$$\tilde{h}_2 = u_1^{z_{21}}.u_2^{z_{22}}$$

and to check the value of v by

$$v = (g_1)^{x_1+\alpha y_1}(g_2)^{x_2+\alpha y_2}$$

with $\alpha = H_1(s, u_1, u_2)$. From there, one can decrypt. However, it is not possible to perform the crucial check $u_2 = (u_1)^w$, since w is missing. Thus, there is a risk that $\mathcal{B}$ fails to reject an invalid ciphertext such that $u_2 \neq (u_1)^w$.

At the end of the simulation, $\mathcal{B}$ returns a triple $(s, u_1, u_2) \neq (s', u_1', u_2')$ such that $H_1(s, u_1, u_2) = H_1(s', u_1', u_2')$ if one has been queried and next returns the output b' of $\mathcal{A}$.

We wish to compare the behaviour of several games

1. game $\mathcal{G}_1$, where (g_1, g_2, u_1', u_2') is from $\mathbf{D}$ and the decryption queries are answered by an actual decryption oracle

2. game $\mathcal{G}_2$, where (g_1, g_2, u_1', u_2') is from $\mathbf{D}$ and the decryption queries are answered by the simulator $\mathcal{B}$

3. game $\mathcal{G}_3$, which is as $\mathcal{G}_2$ but stops as soon as it has found a collision

4. game $\mathcal{G}_4$, which is as $\mathcal{G}_3$ but where (g_1, g_2, u_1', u_2') is from $\mathbf{R}$

We observe that the probability that $b' = b$ in game $\mathcal{G}_1$ is $\frac{1}{2}(\theta_1 + (1 - \theta_0))$, where θ_0 be the probability that algorithm $\mathcal{A}$ outputs 1 on inputs taken from $\mathcal{D}_0$ and θ_1 the probability that it outputs 1 when they are taken from $\mathcal{D}_1$. This is $1/2 + \frac{\varepsilon}{2}$. We check that the probabilities of the same event in all games are very close to each other repeatedly using the simple yet useful lemma from [27]

Lemma 1 *Let E, F, and E', F' be events of two probability spaces such that both*

$$\Pr[E|\neg F] = \Pr[E'|\neg F'] \text{ and } \Pr[F] = \Pr[F'] \leq \varepsilon.$$

Then,

$$|\Pr[E] - \Pr[E']| \leq \varepsilon$$

Proof: We write

$$\Pr[E] = \Pr[E|\neg F]\Pr[\neg F] + \Pr[E|F]\Pr[F]$$
$$\Pr[E'] = \Pr[E'|\neg F']\Pr[\neg F'] + \Pr[E|F']\Pr[F']$$

Hence

$$\Pr[E] - \Pr[E'] = \Pr[E|\neg F](\Pr[\neg F] - \Pr[\neg F']) + (\Pr[E|F]\Pr[F] - \Pr[E|F']\Pr[F'])$$

The right hand side becomes $\Pr[E|F]\Pr[F] - \Pr[E|F']\Pr[F']$, which is bounded by ε.

We see that $\mathcal{G}_2$ perfectly simulates $\mathcal{A}$ unless an invalid ciphertext is queried and not rejected. Let us interpret the event in terms of the secret parameters x_1, x_2, y_1, y_2. These parameters are in a two-dimensional plane $\mathcal{P}$ defined in terms of the unknown logarithm w of g_2 in base g_1 by equations:

$$\log c = x_1 + wx_2$$

$$\log d = y_1 + wy_2$$

An invalid ciphertext with preamble (s, u_1, u_2, v) gets accepted if

$$\log v = r(x_1 + \alpha y_1) + wr'(x_2 + \alpha y_2)$$

with $r = \log u_1$, $r' = \log_{g_2} u_2$, $r \neq r'$. This additional equation is not a linear combination of the above two. Thus, it defines a line within $\mathcal{P}$. Since κ ciphertexts are queried, we can collect the exceptional lines as a subset of probability $\leq \frac{\kappa}{q}$. Applying the lemma we see that $\mathcal{G}_1$ and $\mathcal{G}_2$ output a correct guess with probabilities that differ by at most $\frac{\kappa}{q}$.

To go from $\mathcal{G}_2$ to $\mathcal{G}_3$, we just have to exclude the event of probability ε_H that a collision is output. This bounds the difference of the probabilities that each game outputs one.

To go from $\mathcal{G}_3$ to $\mathcal{G}_4$, we consider the event that either

- $\log_{g_1} u'_1 = \log_{g_2} u'_2$
- or some invalid ciphertext with $(s, u_1, u_2) \neq (s', u'_1, u'_2)$ is accepted

Let us interpret these events in terms of the secret parameters x_1, x_2, y_1, y_2. If $\log_{g_1} u'_1 \neq \log_{g_2} u'_2$ these parameters are further constrained by equation

$$\log v = r_1(x_1 + \alpha' y_1) + wr'_1(x_2 + \alpha y_2)$$

with $r = \log u_1$, $r'_1 = \log_{g_2} u_2$, $r_1 \neq r'_1$. This defines a line $\mathcal{L}$. Now an invalid ciphertext with preamble (s, u_1, u_2, v) gets accepted if

$$\log v = r(x_1 + \alpha y_1) + wr'(x_2 + \alpha y_2)$$

with $r = \log_{g_1} u_1$, $r' = \log_{g_2} u_2$, $r \neq r'$. Since no collision has been found by $\mathcal{B}$, $\alpha \neq \alpha'$ and this additional equation is not a linear combination of the above three. Thus, it defines a point within $\mathcal{L}$. Since κ ciphertexts are queried, we can collect the exceptional points as a subset of probability $\leq \frac{\kappa}{q}$. We add the probability $1/q$ that $\log_{g_1} u'_1 = \log_{g_2} u'_2$. We consider a similar event for $\mathcal{G}_4$, which we also discard. Applying the lemma we see that $\mathcal{G}_3$ and $\mathcal{G}_4$ output a correct guess with probabilities which differ by at most $\frac{\kappa+1}{q}$.

To conclude, observe that, when (g_1, g_2, u'_1, u'_2) is from $\mathbf{R}$, the pair γ_1, γ_2 is independent of (g_1, g_2, u'_1, u'_2) and thus the distribution of inputs for game $\mathcal{G}_4$ is independent of b. Accordingly, the probability that the algorithm outputs b is $1/2$. Summing up, we get that $\frac{\varepsilon}{2} \leq \varepsilon_{DDH} + \varepsilon_H + \frac{2\kappa+1}{q}$. This finishes the proof of the theorem.

4.3 Adaptive chosen-ciphertext security

We prove the following:

Theorem 3 *Let $\mathcal{A}$ be a CCA–adversary $\mathcal{A}$ attacking $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, within time bound t, with advantage ε, making κ queries to the decryption oracle and using two test messages with ℓ bytes. There exists machines $\mathcal{B}_{DDH}$, $\mathcal{B}_H$, $\mathcal{B}_M$ and $\mathcal{B}'_H$ operating within time bound t', such that $\mathcal{B}_{DDH}$ attacks the DDH with advantage ε_{DDH}, $\mathcal{B}_H$ plays the UWO game for H_1 with success probability ε_H, $\mathcal{B}_M$ attacks the pseudorandomness of MARS in sum/counter mode when used to output the number of blocks requested to encrypt ℓ-byte messages, with advantage ε_M, $\mathcal{B}'_H$ plays the UWO game for H_{31}, with advantage ε'_H, and*

$$\begin{aligned} \varepsilon &\geq \varepsilon_{DDH} + \varepsilon_H + 2\varepsilon_M + \lceil \frac{\ell}{1024} \rceil \varepsilon'_H + \frac{2\kappa+1}{q} + \frac{2}{2^{128}} + \frac{\kappa}{2^{128}} \\ t' &\leq t + \tau \end{aligned}$$

proof: We define the following CCA–attacker $\mathcal{B}$.

1. $\mathcal{B}$ is given its input (g_1, g_2, u'_1, u'_2)
2. $\mathcal{B}$ generates an incomplete secret key x_1, x_2, y_1, y_2, z_{11}, z_{12}, z_{21}, z_{22}
3. $\mathcal{B}$ generates a random s', a random key $k1$, and forms

$$(s', u'_1, u'_2, v', \gamma_1, \gamma_2)$$

where v' is computed by means of the relations

$$v' = (g_1)^{x_1 + \alpha' y_1} (g_2)^{x_2 + \alpha' y_2}$$

γ_1, γ_2 are defined by

$$\gamma_1 = u_1^{z_{11}}.u_2^{z_{12}}$$
$$\gamma_2 = u_1^{z_{21}}.u_2^{z_{22}}$$

4. next $\mathcal{B}$ generates k_2 and runs A_1 on the public data to get a pair of messages $\{M_0, M_1\}$ as well as a state information st. It chooses a random bit b, and then defines $C \leftarrow y$, to be a ciphertext of M_b with preamble (s', u'_1, u'_2, v'), computing the session key k' as $k' = H_2(s', u'_1, \gamma_1, \gamma_2)$

5. $\mathcal{B}$ runs $A_2(C, st)$ and finally gets an answer b'. Finally, $\mathcal{B}$ outputs bit $b = b'$.

As in the proof of theorem 2, we will envision several games:

- game $\mathcal{G}_1$, where (g_1, g_2, u'_1, u'_2) is from $\mathbf{D}$ and the decryption queries are answered by an actual decryption oracle
- game $\mathcal{G}_2$, where (g_1, g_2, u'_1, u'_2) is from $\mathbf{D}$ and the decryption queries are answered by the simulator used in the proof of theorem 2
- game $\mathcal{G}_3$, which is as $\mathcal{G}_2$ but stops as soon as it has found a collision for H_1
- game $\mathcal{G}_4$, which is as $\mathcal{G}_3$ but where (g_1, g_2, u'_1, u'_2) is from $\mathbf{R}$. As already observed, this makes the pair γ_1, γ_2 random
- game $\mathcal{G}_5$, which plays $\mathcal{G}_4$ but uses a randomly chosen key k' in place of $k' = H_2(s', u'_1, \gamma_1, \gamma_2)$ to compute the challenge ciphertext. Also, when a ciphertext is queried with preamble (s', u'_1, u'_2, v'), the same key k' is used to decrypt.
- game $\mathcal{G}_6$, which plays $\mathcal{G}_5$ but rejects when a ciphertext with preamble (s', u'_1, u'_2, v') is submitted
- game $\mathcal{G}_7$, which is as $\mathcal{G}_6$ but replaces the output of the MARS pseudo-random generator produced from preamble (s', u'_1, u'_2, v') by a random string of the length appropriate for decrypting the challenge ciphertext.

We observe that the probability that $\mathcal{G}_1$ outputs 1 is exactly $1/2 + \varepsilon$. As in the proof of theorem 2, we bound the difference of probabilities between $\mathcal{G}_1$ and $\mathcal{G}_4$ by $\varepsilon_{DDH} + \varepsilon_H + \frac{2\kappa+1}{q}$. Clearly, the output of $\mathcal{G}_7$ is independent of b. Thus, we only have to upperbound the difference that the output is 1 in games $\mathcal{G}_4$ and $\mathcal{G}_7$.

We first study how one goes from $\mathcal{G}_4$ to $\mathcal{G}_5$. We use the fact that H_2 is *universal* in terms of the inputs $\tilde{h}_1, \tilde{h}_2$: for every pair x, y of such inputs, $x \neq y$, the probability (over the key k_2) that $H_2(x) = H_2(y)$ is $1/2^\ell$, where $\ell = 256$ is the size of the outputs of H_2. When this holds, the leftover hash lemma of [16] implies that, hashing a set of 2^λ bit strings produces a distribution $(k_2, H_2(x))$, whose distance to the uniform distribution is $\leq \frac{1}{2^{(\lambda-\ell)/2}}$. Here, $\lambda \geq 2 \times 255$ and therefore the distance is at most $2/2^{128}$. This bounds the difference between the respective probabilities that $\mathcal{G}_4$ and $\mathcal{G}_5$ output 1.

We next bound the difference of probabilities from $\mathcal{G}_5$ to $\mathcal{G}_6$. We have to consider the probability that $\mathcal{G}_5$ does not reject a ciphertext with preamble (s', u_1', u_2', v'). We distinguish between

- ciphertexts queried by A_1
- ciphertexts queried by A_2, (which are different from the challenge ciphertext)

Game $\mathcal{G}_5$ perfectly simulates $\mathcal{G}_4$ unless a queried ciphertext with preamble (s', u_1', u_2', v') is correctly MACed. In this case

- ciphertexts queried by A_1 have their first block correctly MACed
- ciphertexts queried by A_2, have the first block on which they differ from the challenge ciphertext correctly MACed

We let Y be the union of these events. In order to bound the probability of Y, we modify game $\mathcal{G}_5$ by replacing the output of the MARS pseudo-random generator produced from preamble (s', u_1', u_2', v') (more accurately from key k') by a random string, whose length is exactly what is needed for decrypting the challenge ciphertext (if additional random data is needed, we simply let MARS output these data). We let ε_{M1} be the distinguishing probability (with respect to testing whether Y happens) between $\mathcal{G}_5$ and $\mathcal{G}_5^{(1)}$. Note that these are two versions of the same machine $\mathcal{B}_{M1}$, one with inputs produced by MARS and the other with random inputs. Similarly, we replace $\mathcal{G}_5^{(1)}$ by $\mathcal{G}_5^{(2)}$, which is alike but stops whenever the various computations of H_{31} needed to check the MACs of the queried ciphextexts, produce a second preimage of some hash value previously obtained when computing the challenge ciphertext. The resulting game $\mathcal{G}_5^{(2)}$ behaves as $\mathcal{G}_5^{(1)}$ (with respect to testing whether Y happens), except on a set of probability bounded by ε_H, where ε_H is the probability that $\mathcal{G}_5^{(2)}$ has won one of $\lceil \frac{\ell}{1024} \rceil$ UOW games for H_{31}. This comes from the fact that $\mathcal{B}_H = \mathcal{G}_5^{(2)}$ stops earlier if this happens. Now, in game $\mathcal{G}_5^{(2)}$, all inputs submitted to the universal hash function H_{32} are distinct from those submitted while encrypting the challenge ciphertext. By standard results (see [19]), each is correct with probability $\leq \frac{1}{2^{128}}$. This shows that $\Pr[Y] \leq \varepsilon_{M1} + \lceil \frac{\ell}{1024} \rceil \varepsilon_H' + \frac{\kappa}{2^{128}}$.

To complete the proof of the theorem, we simply note that $\mathcal{G}_6$ and $\mathcal{G}_7$ are two versions of the same machine $\mathcal{B}_{M2}$, one with inputs produced by MARS and the other with random inputs. We let ε_M be the distinguishing probability of the machine defined by running $\mathcal{B}_{M1}$ and $\mathcal{B}_{M2}$ at random, we see that:

$$\begin{aligned} \varepsilon &\geq \varepsilon_{DDH} + \varepsilon_H + \varepsilon_{M1} + \varepsilon_{M2} + \lceil \frac{\ell}{1024} \rceil \varepsilon_H' + \frac{2\kappa+1}{q} + \frac{2}{2^{128}} + \frac{\kappa}{2^{128}} \\ \varepsilon &\geq \varepsilon_{DDH} + \varepsilon_H + 2\varepsilon_M + \lceil \frac{\ell}{1024} \rceil \varepsilon_H' + \frac{2\kappa+1}{q} + \frac{2}{2^{128}} + \frac{\kappa}{2^{128}} \end{aligned}$$

This concludes the proof.

4.4 Construction of the basic primitives

From the security analysis that was just performed, one sees that it is necessary to ensure that the hash functions have the required properties.

4.4.1 Construction of H_1 and H_{31}

Both functions use the construction in [25]. This construction allows to use a fixed-size compression function which compresses $a + b$ bits to b bits in order to hash messages with La blocks. It has the following properties

1. if the original compression function is second preimage resistant, then, the resulting function is OWU (one-way universal)

2. the key to the composed function consists of a a-bit string K, together with $\ell + 1$ b-bit masks m_i, where $\ell = \lceil \log L \rceil$

Recall that a hash function H is OWU, if it is hard for an adversary $\mathcal{A}$ to win the following game:

1. $\mathcal{A}$ chooses a message x

2. $\mathcal{A}$ receives a random key K for H

3. $\mathcal{A}$ outputs a message y and wins if it has found a collision, i.e. $y \neq x$ and $H(K, y) = H(K, x)$.

As usual, one can define the advantage of $\mathcal{A}$ as its probability of success.

The construction of [25] hashes x, a message with L blocks x_i, by computing h_i, where h_0 is an arbitrary string and $h_i = C(h_{i-1} \oplus m_{\nu(i)}, x_i \oplus K)$. The ν function computes the exponent $\nu(i)$ of 2 in the prime decomposition of i.

We do not review the analysis from [25] but mention that the existence of an adversary $\mathcal{A}$ that wins the OWU game with advantage ε, within time t, implies the existence of an adversary $\mathcal{B}$ which computes preimages with probability ε' and time t', where $t' \simeq t$ and $\varepsilon' \simeq \frac{\varepsilon}{\ell}$.

The cryptosystem under review uses the above construction, taking as a compression function the core function of $SHA1$. This means $a = 512$ and $b = 160$. We do not discuss the necessary padding rules. The requested key material comes from the output of MARS. This defines what is denoted by $UOWHash$ in [27] and H_{31} in our analysis. What we call H_1 ($UOWHFhash'$ in [27]) applies H_{31} to a specific formatting of the data s, u_1, u_2.

4.4.2 Construction of H_2

Hash function H_2 is the *entropy smoothing* hash function that produces a random key k for MARS. It consists of two components that are xored together:

1. a 256-bit string obtained by applying a universal hash fonction to formatted data coming from the pair $(\tilde{h}_1, \tilde{h}_2)$. This uses a standard construction of a universal hash function
$$\sum_{i=1}^{\ell} k_i m_i$$
where ℓ is an appropriate parameter, k_i comes form the key and m_i are the message blocks. The operations are performed in the finite field $GF(2^{256})$.

2. a 256-bit string obtained by applying (twice) a simplified SHA function to suitably formatted data coming from $s, u_1, \tilde{h}_1, \tilde{h}_2$

Note that the first component should be enough. The role of the second component will be explained later.

4.4.3 Construction of H_3

As explained earlier in our report, H_3 is the composition of H_{31} (*UOWHash'* of [27]), which produces a 160-bit output, followed by a universal hash function in the sense of Carter and Wegman (see [8] and also [19]). The latter uses the formula $c_1 d_1 + c_2 d_2$ to hash c_1, c_2 with key d_1, d_2, where computations are in the field $GF(2^{128})$. As usual, the key comes from the material produced by the output of the MARS generator.

4.5 Hash functions as a hedge

In the random oracle model [2], one can prove that the scheme achieves IND-CCA security relative to the CDH problem. More accurately, one can use a version of the random oracle model which assumes that the core compression function of $SHA1$ behaves like a random function. Looking at the construction of the entropy-smoothing hash function H_2, one sees that, due to its second component, it also behaves like a random function. Therefore, one can simply see H_2 as a random oracle.

We will not review in detail the proof in [26]. We mention that it uses a reduction where one is given access to a DDH distinguisher $\mathcal{B}$, whose advantage is $1 - \delta$, with δ negligible. From an adversary $\mathcal{A} = (A_1, A_2)$ against the semantic security of $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, one gets another machine $\mathcal{S}$ which solves the CDH, as follows:

1. $\mathcal{S}$ is given a CDH instance, (g, A, B) in $\mathbb{Z}_P^\star$ with g of order q, and has to find the CDH answer C. It first simulates $\mathcal{K}(1^m)$ by choosing random $w, x, y, z_2 \in \mathbb{Z}_q$,

random keys k_1, k_2, and defining

$$g_1 \leftarrow g, h_1 \leftarrow A, g_2 \leftarrow g^w \bmod P, h_2 \leftarrow g^{z_2} \bmod P, c \leftarrow g^x \bmod P, d \leftarrow g^y \bmod P.$$

2. $\mathcal{S}$ runs A_1 on the public data, and gets a pair of messages $\{M_0, M_1\}$ as well as a state information st. It chooses a random bit b, and then defines, for a random 128-bit string s',

$$u'_1 \leftarrow B, u'_2 \leftarrow u'^{w}_1 \bmod P, \alpha' = H_1(s', u'_1, u'_2), v' = u'^{x+\alpha' y}_1 \bmod P.$$

   Then it chooses a random 256-bit key k', and encrypts the message M_b into e', using key k'. It produces the ciphertext $C' = (s', u'_1, u'_2, v', e')$. (The dash is used for denoting the target ciphertext.)

3. $\mathcal{S}$ runs $A_2(C', st)$ and finally gets an answer b'. Then $\mathcal{S}$ outputs the solution to the CDH instance found during the simulation as explained below, or Fail.

In order to simulate, the random oracle, $\mathcal{S}$ can make random choices unless it meets data $(s', u'_1, \tilde{h}'_1, \tilde{h}'_2)$ corresponding to the challenge ciphertext. Even though $\tilde{h}'_1$ is unknown, this can be checked, with low error probability by calling the DDH distinguisher. One can show that it happens with significant probability and produces the requested CDH result.

To consider $\mathcal{A}$ as a chosen-ciphertext adversary, $\mathcal{S}$ has also to be able to simulate the decryption oracle. In order to find the actual symmetric key k that is needed to decrypt $C = (s, u_1, u_2, v, e)$, $\mathcal{S}$ looks at the list of queries to H_2 and checks whether it finds one of the form $(s, u_1, x, \tilde{h}_2)$. Note that $\tilde{h}_2$ can be computed from the preamble and z_2. The correctness of x is checked by running the DDH distinguisher on (g_1, h_1, u_1, x).

One can see that the reduction calls $\mathcal{B}$ as many times as there are queries to the random oracle H_2. Now, one can substitute to $\mathcal{B}$ the machine that is obtained through the reduction which comes from an adversary against the security of the scheme in the standard model. However, as observed in section 3.1.1, it is necessary to run $O(\mathtt{AdvDDH}^{-2})$ tests to get a negligible error probability. Thus, the concrete estimates that may be derived from the proof in the random oracle model are not very conclusive.

4.6 Understanding the rationale for the construction

As pointed out in section 2.2, the security proof is intricate and the rationale for each construction is only apparent to cryptography experts, who have mastered the entire sequence of arguments. To support this opinion, we answer the three questions raised in section 2.2.

1. Document [27] introduces two values h_1, h_2 whereas the research paper [26] on which it builds has only one such value h to guarantee that enough randomness

is given to H_2 so that 256 bits are obtained by entropy smoothing. With a single h, the current security proof collapses.

2. If one derives the symmetric encryption key k by applying $SHA1$ to the data s, u_1, $\tilde{h_1}$, $\tilde{h_2}$ instead of using the function proposed in the specification, there is no guarantee that the session key is random enough. The current security proof, in the standard model, collapses at the point where one goes from from $\mathcal{G}_4$ to $\mathcal{G}_5$ in the proof of theorem 3.

3. If one substitutes another symmetric encryption scheme to the one specifically described in the specification, the security proof may or may not collapse. Replacing MARS by the AES is acceptable but discarding the MACs is not, since the current security proof collapses at the point where one goes from from $\mathcal{G}_5$ to $\mathcal{G}_6$ in the proof of theorem 3

5 Conclusion

Based on our analysis, we believe that the cryptosystem ACE is provably secure against adaptive chosen-ciphertext attacks. The proposed parameters appear to offer a margin of security of at least fifty years. We therefore recommend the scheme.

Our main concern is the lack of flexibility of the scheme. As explained in our report, it is extremely difficult to "reengineer" any part of the scheme, due to the intricate security arguments.

Another warning that we give is that the scheme relies on the hardness of the Diffie-Hellman decisional hypothesis DDH. Thus, the comparison with other schemes based –say– on the computational Diffie-Hellman hypothesis is a bit unfair.

To end up, we would like to emphasis that the achievement of [27], providing a complete quantitative security analysis from standard hypotheses (the only such example, as far as we know) is a real masterpiece.

References

[1] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. In *Crypto '98*, LNCS 1462, pages 26–45. Springer-Verlag, Berlin, 1998.

[2] M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols. In *Proc. of the 1st CCS*, pages 62–73. ACM Press, New York, 1993.

[3] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92–111. Springer-Verlag, Berlin, 1995.

[4] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. C. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, P. Zimmermann, Factorization of a 512-Bit RSA Modulus. Eurocrypt'2000, Lecture Notes in Computer Science 1807,(2000), 1–18.

[5] Certicom, Information on the Certicom ECC challenge, `http://www.certicom.com/research/ecc_challenge.html`

[6] D. Coppersmith, A.M. Odlyzko and R.Schroeppel, Discrete Logarithms in GF(p), *Algorithmica 1*(1986), 1–15

[7] R. Cramer and V. Shoup, A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. Crypto'98, Lecture Notes in Computer Science 1462, (1998), 13–25.

[8] J.L. Carter and M.N. Wegman, Universal classes of hash functions, *Journal of Computer and System Sciences*, 18, (1979), 143–154.

[9] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *Proc. of the 23rd STOC*. ACM Press, New York, 1991.

[10] T. El Gamal, A public key crtyptosystem and signature scheme based on discrete logarithms, *IEEE Trans. on Inform. theory*, 31 (1985), 469–472.

[11] G. Frey, M. Müller, and H. G. Rück. The Tate-Pairing and the Discrete Logarithm Applied to Elliptic Curve Cryptosystems. *IEEE Transactions on Information Theory*, 45:1717–1719, 1999.

[12] G. Frey and H. G. Rück. A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, 62:865–874, 1994.

[13] S. Goldwasser and S. Micali, Probabilistic encryption, *Journal of Computer and System Science* 28, (1984), 270–299.

[14] Dan Gordon, Discrete Logarithms in $GF(p)$ using the Number Field Sieve, *SIAM J. Discrete Math.*, 6, (1993), 124-138.

[15] R. Harley, D. Doligez, D. de Rauglaudre, X. Leroy, Elliptic Curve Discrete Logarithms: ECC2K-108,
`http://cristal.inria.fr/ harley/ecdl7/`

[16] R. Impagliazzo and D. Zuckermann, How to rectcle random bits, *30th annual symposium on foundations of computer science*, (1989), 248–253.

[17] A. Joux and R. Lercier, Computing a discrete logarithm in GF(p), p a 90 digit prime,
`http://www.medicis.polytechnique.fr/ lercier/english/dlog.html`

[18] A. Joux and R. Lercier, Computing a discrete logarithm in GF(p), p a 100 digit prime,
`http://www.medicis.polytechnique.fr/ lercier/english/dlog.html`

[19] H. Krawczyk, LFSR-based hashing and authentication, Crypto'94, Lecture Notes in Computer Science 839, (1995), 129–139.

[20] A.K. Lenstra and E. Verheul, Selecting cryptographic key sizes, PKC'2000, Lecture Notes in Computer Science 1751,(2000), 446–465.

[21] M. Naor, O. Reingold, Number-theoretic Constructions of Efficient Pseudo-random Functions, *38-th annual symposium on foundations of computer science*, (1997), 458–467.

[22] S. Pohlig and M. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Transactions on Information Theory*, 24, (1978),106–110.

[23] J. Pollard, Monte Carlo methods for index computation mod p, *Mathematics of Computation*, 32, (1978), 918–924.

[24] Oliver Schirokauer, Discrete Logarithms and Local Units, *Phil. Trans. R. Soc. Lond.* A 345, (1993), 409–423.

[25] V. Shoup, A composition theorem for universal one-way hash functions, Eurocrypt'2000, Lecture Notes in Computer Science 1807, (2000), 275–288.

[26] V. Shoup, Using hash functions as a hedge against chosen ciphertext attack, Eurocrypt'2000, Lecture Notes in Computer Science 1807, (2000), 445–452.

[27] V. Shoup and T. Schweinberger, ACE Encrypt: The Advanced Cryptographic Engine' public key encryption scheme, Manuscript, March 2000. Revised, August 14, 2000.

[28] P.C. van Oorschot and M. J. Wiener, Parallel collision search with cryptanalytic applications, *J. Cryptology*, 12, (1999), 1–28.

[29] D. Weber, T. F. Denny and J. Zayer,
`http://felix.unife.it/Root/d-Mathematics/d-Number-theory`
`/t-Weber-discrete-logarithm-record-960925`