Fast Cramer-Shoup Cryptosystem

Anonymized for submission

Keywords: Public Key Encryption, Cramer-Shoup, IND-CCA2.

Abstract: Cramer-Shoup was the first practical adaptive CCA-secure public key encryption scheme. We propose a faster version of this encryption scheme, called *Fast Cramer-Shoup*. We show empirically and theoretically that our scheme is faster than three versions proposed by Cramer-Shoup in 1998. We observe an average gain of 60% for the decryption algorithm. We prove the IND-CCA2 security of our scheme. The proof only relies on intractability assumptions like DDH.

1 Introduction

Provable security is an important issue in modern cryptography. It allows us to formally prove the security of the encryption schemes by reduction to difficult problems such as discrete logarithm problem (DL), Computational Decisional Diffie-Hellman problem (CDH), Decision Diffie-Hellman problem (DDH) (Boneh, 1998; Joux and Guyen, 2006) or the quadratic residuosity problem. For instance, the DDH problem is used to prove the IND-CPA security of the ElGamal encryption scheme (Elgamal, 1985). In order to have security against adaptive chosen ciphertext attacks (IND-CCA2), a notion introduced in 1991 by Dolev et al. (Dolev et al., 1991), Cramer and Shoup proposed in 1998 an encryption scheme (Cramer and Shoup, 1998a) that has a verification mechanism in the decryption algorithm to avoid malleability of the ciphertext and also uses one hash function.

Fujisaki and Okamoto in (Fujisaki and Okamoto, 1999) proposed a generic conversion from any IND-CPA cryptosystem into an IND-CCA2 one, in the random oracle model (ROM) (Bellare and Rogaway, 1993). However the design of an IND-CCA2 encryption scheme is not easy, as the story of Optimal Asymmetric Encryption Padding (OAEP) (Bellare and Rogaway, 1994; Pointcheval, 2011) can show. After a first try by Bellare and Rogaway (Bellare and Rogaway, 1994) in 1995, V. Shoup found a problem in (Shoup, 2001), that was fixed in (Phan and Pointcheval, 2004; Pointcheval, 2011). Finally to conclude the story of OAEP, an computer verified proof has been made in (Barthe et al., 2011).

Our goal is to design a faster version of Cramer-Shoup scheme. For this, we use the approach proposed in (Sow and Sow, 2011) to improve the decryption algorithm of ElGamal (Elgamal, 1985).

Contributions: Our main aim is to improve the efficiency of the Cramer-Shoup public key scheme. Our contributions are as follows:

1. We design a public key cryptosystem, called *Fast Cramer-Shoup*, based on the Generalized ElGamal encryption scheme (Sow and Sow, 2011). We follow the spirit of Cramer-Shoup versions introduced by Cramer-Shoup in (Cramer and Shoup, 1998b; Cramer and Shoup, 1998a), but we modify the key generation and the decryption algorithm in order to be faster.
2. We implement all these schemes with GMP (Granlund and the GMP development team, 2020) to demonstrate that Fast Cramer-Shoup is the fastest one. Our experiments show that we have a gain of 60% for decryption algorithm compared to the most efficient version proposed by Cramer Shoup. This has an important impact because Cramer-Shoup is used in the standard ACE-KEM of ISO/IEC 18033-2:2006 (International Organization for Standardization, 2006).
3. We prove its security against the adaptive chosen ciphertext attack (IND-CCA2) under the (DDH) assumption.

Related works: Shoup and Gennaro (Shoup and Gennaro, 1998) give two ElGamal-like practical threshold cryptosystems that are secure against adaptive chosen ciphertext attack in the random oracle model. They use $H(h^r) \oplus m$ to encrypt the message m, unfortunately the trick of Sow et al. (Sow and Sow, 2011) cannot be applied in this case.

In (Cramer and Shoup, 2002), the authors proposed a construction by considering an algebraic primitive called *universal hash proof systems*. They showed that this framework yields not only the original DDH-based Cramer-Shoup's scheme but also encryption schemes based on quadratic residuosity and on Paillier's assumption (Paillier, 1999).

In 2011, a modified variant of ElGamal's encryption scheme was presented (Sow and Sow, 2011), and it is called *Generalized ElGamal's* encryption scheme. This version is faster than ElGamal because the decryption key size is smaller. More precisely, the encryption algorithm has the same efficiency than ElGamal's encryption mechanism, the key generation algorithm is slower but the decryption process is faster. This is not a problem since the key generation is done only once. We adapt this idea to improve Cramer and Shoup's encryption scheme.

Outline: In Section 2, we present three versions of Cramer-Shoup's encryption scheme. In Section 3, we propose our public key cryptosystem, called *Fast Cramer-Shoup*. In Section 4, we present the result of our empirically performance comparison and our complexity analysis for the key generation, encryption and decryption algorithms for all versions of Cramer-Shoup.

2 Cramer-Shoup's Encryption Schemes

We recall the original Cramer and Shoup's encryption scheme presented in the eprint version (Cramer and Shoup, 1998b), then the standard Cramer-Shoup's version published in CRYPTO'98 (Cramer and Shoup, 1998a) and finally the efficient Cramer-Shoup's version also proposed in (Cramer and Shoup, 1998b).

All these schemes are composed of a key generation algorithm, an encryption and a decryption algorithm. The decryption algorithm consists of two algorithms one for recovering from the ciphertext the plaintext and one to check the non-malleability of the ciphertext in order to ensure IND-CCA2 security.

2.1 Original Cramer-Shoup's Encryption

This scheme, that we call *Original Cramer-Shoup*, was published on March 4th, 1998 on IACR eprint (Cramer and Shoup, 1998b), only few days after the deadline of CRYPTO'98, on February 16, 1998 and works as follows.

Original Key Generation Algorithm:
1. Select a group G of prime order q.
2. Choose eight random elements: $g_1, g_2 \in G$ and $x_1, x_2, y_1, y_2, z_1, z_2 \in \mathbb{Z}_q$.
3. Compute in G: $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}, h = g_1^{z_1} g_2^{z_2}$.
4. Choose H that hashes messages to elements of $\mathbb{Z}_q$.
5. Return $pk = (g_1, g_2, c, d, h, H)$ and $sk = (x_1, x_2, y_1, y_2, z_1, z_2)$.

Original Encryption Algorithm:
1. Choose a random element $r \in \mathbb{Z}_q$.
2. Compute $u_1 = g_1^r, u_2 = g_2^r, e = h^r m, \alpha = H(u_1, u_2, e)$ and $v = c^r d^{r\alpha}$.
3. Return the following ciphertext: (u_1, u_2, e, v).

Original Decryption Algorithm:
1. Compute $\alpha = H(u_1, u_2, e)$.
2. Verify if $u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} = v$.
3. Output $m = e u_1^{-z_1} u_2^{-z_2}$ if the condition holds, otherwise output "reject".

2.2 Standard Cramer-Shoup's Encryption

We call the version published at CRYPTO'98 the *Standard Cramer-Shoup* version. The main difference is that they use only one z instead of z_1 and z_2.

Standard Key Generation Algorithm:
1. Select a group G of prime order q.
2. Choose randomly $g_1, g_2 \in G$; $x_1, x_2, y_1, y_2, z \in \mathbb{Z}_q$.
3. Compute in G: $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$ and $h = g_1^z$.
4. Choose H that hashes messages to elements of $\mathbb{Z}_q$.
5. Return (pk, sk) where $pk = (g_1, g_2, c, d, h, H)$ and $sk = (x_1, x_2, y_1, y_2, z)$.

Standard Decryption Algorithm:
1. Compute $\alpha = H(u_1, u_2, e)$.
2. Verify if $u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} = v$.
3. Output $m = e u_1^{-z}$ if 2 holds, otherwise reject.

2.3 Efficient Cramer-Shoup Encryption

In (Cramer and Shoup, 1998b), authors propose a "*somewhat more efficient variant of the scheme*". Since encryption algorithms are identical, we just present key generation and decryption algorithms.

Efficient Key Generation Algorithm:
1. Select a group G of prime order q.
2. Choose randomly $g_1 \in G$ and $w, x, y, z \in \mathbb{Z}_q, w \neq 0$.
3. Compute $g_2 = g_1^w, c = g_1^x, d = g_1^y$ and $h = g_1^z$ in G.
4. Choose H that hashes messages to elements of $\mathbb{Z}_q$.
5. Return $pk = (g_1, g_2, c, d, h, H)$ and $sk = (w, x, y, z)$.

Efficient Decryption Algorithm:

1. Compute $\alpha = H(u_1, u_2, e)$.
2. Verify if $u_1^w = u_2$ and $u_1^{x+y\alpha} = v$.
3. Output $m = eu_1^{-z}$ if 2 holds, otherwise reject.

3 Fast Cramer-Shoup's Encryption Scheme

We present our Fast Cramer-Shoup's scheme then we prove that it is IND-CCA2 secure under DDH assumption and particular hash function. Let us recall some notions and definitions.

3.1 Notations and definitions

Some notions and definitions like the set of non-negative integers $\mathbf{Z}_{\geq 0}$, a security parameter λ, a group description Γ, a computational group scheme $\mathcal{G}$, a probability distribution of group descriptions S_λ, hash functions (**HF**), target collision resistant (TCR) assumption for hash function (**HF**), some random variables as **Coins** used in the following are defined in (Cramer and Shoup, 2003) (see also (Naor and Yung, 1989)). In (Cramer and Shoup, 2003),the original Cramer-Shoup cryptosystem is called **CS1**.

- A computational group scheme $\mathcal{G}$ specifies a sequence $(S_\lambda)_{\lambda \in \mathbf{Z}_{\geq 0}}$ of group distributions. For every value of a security parameter $\lambda \in \mathbf{Z}_{\geq 0}$, S_λ is a probability distribution of group descriptions.
- A group description Γ specifies a finite abelian group $\hat{G}$, along with a prime-order subgroup G, a generator g_2 of G, and the order q of G. We use multiplicative notation for the group operation in $\hat{G}$, and we denote the identity element of $\hat{G}$ by 1_G.
- We denote $\Gamma[\hat{G}, G, g_2, q]$ so that Γ specifies $\hat{G}$, G, g_2, and q as above.

3.2 Description of Fast Cramer-Shoup's Scheme

Our Fast Cramer-Shoup's encryption scheme contains a key generation algorithm, an encryption and a decryption algorithm. It relies on the fact that we can select adapted random in order to have a faster decryption algorithm.

Fast Key Generation Algorithm:

G1 : On input 1^λ for $\lambda \in \mathbf{Z}_{\geq 0}$, select a group $\hat{G}$, along with a prime-order subgroup G and choose a generator $g_2 \in G$ of order q. So, $\Gamma[\hat{G}, G, g_2, q] \overset{R}{\leftarrow} S(1^\lambda)$;

G2 : Pick random elements $x, y, k, t \in \mathbb{Z}_q$ with $log_2(t) = \frac{log_2(q)}{2}$, and compute $w', z \in \mathbb{Z}_q$ such that $kq = tw' + z$ and then compute $w \equiv w'$.

G3 : Compute $g_1 = g_2^w, c = g_2^{wx}, d = g_2^{wy}$ and $h = g_2^z$.

G4 : Choose a hash function $H \overset{R}{\leftarrow} \mathbf{HF}$.

G5 : Return (pk, sk), where $pk = (\Gamma, H, g_1, c, d, h)$ and $sk = (\Gamma, H, t, x, y, z)$.

Remark 3.1. *The size of t is half the size of q. Thus the size of z is smaller or equal to the size of t, i.e., $log_2(z) \leq log_2(t)$.*

Fast Encryption Algorithm: Encrypt a message m with $pk = (\Gamma, H, g_1, c, d, h)$.

E1 : Choose a random element $r \in \mathbb{Z}_q$ and compute,

E2 : $u_1 = g_1^r$;

E3 : $u_2 = g_2^r$;

E4 : $u_3 = h^r$;

E5 : $e = u_3 m$;

E6 : $\alpha = H(u_1, u_2, e)$;

E7 : $v = c^r d^{r\alpha}$ and output the ciphertext $\psi = (u_1, u_2, e, v)$.;

Fast Decryption Algorithm: Decrypt a ciphertext (u_1, u_2, e, v) with $sk = (\Gamma, H, t, x, y, z)$.

D1 : Parse $\psi \leftarrow (u_1, u_2, e, v) \in G^4$; output reject if ψ is not of this form.

D2′ : Test if u_1 and u_2 belong to G; output reject and halt if this is not the case.

D3 : Compute $\alpha = H(u_1, u_2, e)$.

D4′ : Test if $u_1^t u_2^z = 1$ and $v = u_1^{x+y\alpha}$; otherwise output reject and halt.

D5′ : Compute $\beta = u_1^t$.

D6 : Output $m = \beta e$.

Correctness:

Verification: We have $\beta u_2^z = u_1^t u_2^z = (g_1^r)^t g_2^{rz} = (g_2^{wr})^t g_2^{rz} = g_2^{r(tw+z)} = g_2^{rkq} = 1$ since the order of g_2 is q and $u_1^{x+y\alpha} = (g_1^r)^{x+y\alpha} = (g_2^{wr})^{x+y\alpha} = (g_2^{wx})^r (g_2^{wy})^{r\alpha} = c^r d^{r\alpha} = v$.

Decryption: The decryption message is $\beta e = u_1^t e = g_2^{wrt} g_2^{zr} m = g_2^{r(tw+z)} m = g_2^{rkq} m = m$, since the order of g_2 is q.

3.3 Security Proof of Fast Cramer-Shoup Scheme

The proof of Theorem 3.2 is similar to that of the Efficient **CS** denoted **CS1b** in (Cramer and Shoup, 2003). All the games $(\mathbf{G}_0, \cdots, \mathbf{G}_5)$ are described in (Cramer and Shoup, 2003). We will not repeat them in this paper.

As **CS1**'s proof in (Cramer and Shoup, 2003), to prove that Fast Cramer-Shoup (**FCS**) is secure against adaptive chosen ciphertext attack if the DDH assumption holds for $\mathcal{G}$ and the TCR assumption holds for **HF**, we need some notions.

- Suppose PKE is a public-key encryption scheme that uses a group scheme in the following natural way: on input 1^λ, the key generation algorithm runs the sampling algorithm of the group scheme on input 1^λ, yielding a group description Γ.
- For a given probabilistic, polynomial-time oracle query machine $\mathcal{A}, \lambda \in \mathbf{Z}_{\geq 0}$, and group description Γ, let us define $\mathrm{AdvCCA}_{\mathrm{PKE},\mathcal{A}}(\lambda|\Gamma)$ to be $\mathcal{A}$'s advantage in an adaptive chosen ciphertext attack where the key generation algorithm uses the given value of Γ, instead of running the sampling algorithm of the group scheme.
- For all probabilistic, polynomial-time oracle query machines $\mathcal{A}$, for all $\lambda \in \mathbf{Z}_{\geq 0}$, let $Q_{\mathcal{A}}(\lambda)$ be an upper bound on the number of decryption oracle queries made by $\mathcal{A}$ on input 1^λ. We assume that $Q_{\mathcal{A}}(\lambda)$ is a strict bound in the sense that it holds regardless of the probabilistic choices of $\mathcal{A}$, and regardless of the responses to its oracle queries from its environment.

Theorem 3.2. *The Fast Cramer-Shoup is secure against adaptive chosen ciphertext attack if:*

1. *the DDH assumption holds for $\mathcal{G}$;*
2. *and the target collision resistance (TCR) assumption holds for* **HF**.

In particular, for all probabilistic, polynomial-time oracle query machines $\mathcal{A}$, for all $\lambda \in \mathbf{Z}_{\geq 0}$, and all $\Gamma[\hat{G}, G, g_2, q] \in [\mathbf{S}_\lambda]$, we have

$$\left|\mathrm{AdvCCA}_{\mathrm{FCS},\mathcal{A}}(\lambda|\Gamma) - \mathrm{AdvCCA}_{\mathrm{CS1},\mathcal{A}}(\lambda|\Gamma)\right| \leq Q_{\mathcal{A}}(\lambda)/q. \tag{1}$$

Description of games: Suppose that $pk = (\Gamma, H, g_1, c, d, h)$ and $sk = (\Gamma, H, t, x, y, z)$. Let $w = \log_{g_2} g_1$, and define $x, y, z \in \mathbb{Z}_q$ as follows: $x = x_1 + x_2 w$, $y = y_1 + y_2 w$ and $z = z_1 + z_2 w$. We have $x = \log_{g_2^w} c$, $y = \log_{g_2^w} d$, and $z = \log_{g_2} h$.

As a notation convention, whenever a particular ciphertext is under consideration in some context, the following values are also implicitly defined in that context:

- $u_1, u_2, u_3, e, v \in G$ where $\psi = (u_1, u_2, e, v)$ and $u_3 = u_2^z$;
- the random $r \in \mathbb{Z}_q$, where $r = \log_{g_1^w} u_1$.

For the target ciphertext ψ^*, we also denote by $u_1^*, u_2^*, u_3^*, e^*, v^* \in G$ and $r^* \in \mathbb{Z}_q$ the corresponding values. The probability space defining the attack game is then determined by the following, mutually independent, random variables:

- the coin tosses **Coins** of $\mathcal{A}$;
- the values $H, w, x_1, x_2, y_1, y_2, z_1, z_2$ generated by the key generation algorithm;
- the values $\sigma \in \{0,1\}$ and $r^* \in \mathbb{Z}_q$ generated by the encryption oracle.

Let us rewrite the games:

$\mathbf{G}_0$: original attack game, let $\hat{\sigma} \in \{0,1\}$ be the output of $\mathcal{A}$ and T_0 the event $\sigma = \hat{\sigma}$, so $\mathrm{AdvCCA}_{\mathbf{FCS},\mathcal{A}}(\lambda|\Gamma) = |Pr[T_0] - 1/2|$

$\mathbf{G}_1$: We now modify game $\mathbf{G}_0$ to obtain game $\mathbf{G}_1$. These two games are identical, except that instead of using the encryption algorithm as given to compute the target ciphertext ψ^*, we use a modified encryption algorithm, in which steps **E4** and **E7** are replaced by **E4′** $: u_3 = u_2^z$ and **E7′** $: v = u_1^{x+y\alpha}$. The change we have made is purely conceptual. The values of u_3^* and v^* are exactly the same in game $\mathbf{G}_1$ as they were in $\mathbf{G}_0$ so $Pr[T_1] = Pr[T_0]$

$\mathbf{G}_2$: We modify the encryption oracle, replacing step **E3** by **E3′** $: \hat{r} \overset{R}{\leftarrow} \mathbb{Z}_q \setminus \{r\}; u_2 \leftarrow g_2^{\hat{r}}$

Lemma 3.3. *There exists a probabilistic algorithm $\mathcal{A}_1$, whose running time is essentially the same as that of $\mathcal{A}$, such that*

$$|Pr[T_2] - Pr[T_1]| \leq \mathrm{AdvDDH}_{\mathcal{G},\mathcal{A}_1}(\lambda|\Gamma) + 3/q. \tag{2}$$

$\mathbf{G}_3$: We modify the decryption algorithm, replacing steps **D4** and **D5** with **D4′** : Test if $u_1 = u_2^w$ and $v = u_1^{x+y\alpha}$; output reject and halt if this is not the case. **D5′** $: u_3 = u_2^z$. Note that the decryption oracle now make use of w, but does not make use of $x_1, x_2, y_1, y_2, z_1, z_2$, except indirectly through the values x, y, z. Now, let R_3 be the event that in game $\mathbf{G}_3$, some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**. Note that if a ciphertext passes the test in **D4′**, it would also have passed the test in **D4**. It is clear that games $\mathbf{G}_2$ and $\mathbf{G}_3$ proceed identically until the event R_3 occurs. In particular, the events $T_2 \wedge \neg R_3$ and $T_3 \wedge \neg R_3$ are identical. So by difference lemma $|Pr[T_3] - Pr[T_2]| \leq Pr[R_3]$, and so it suffices to bound $Pr[R_3]$. We introduce auxiliary games $\mathbf{G}_4$ and $\mathbf{G}_5$ below to do this.

$\mathbf{G}_4$: We replace step **E5** by **E5′** $: r \overset{R}{\leftarrow} \mathbb{Z}_q; e \leftarrow g^\gamma$ so $Pr[T_4] = 1/2$, since in game $\mathbf{G}_4$, the variable σ is never used. Define the event R_4 to be the event in game $\mathbf{G}_4$ analogous to the event R_3 in game $\mathbf{G}_3$; that is, R_4 is the event that in game $\mathbf{G}_4$, some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**. We show that this modification has no effect; more precisely: $Pr[T_4] = Pr[T_3]$, and $pr[r_4] = pr[r_3]$

$\mathbf{G}_5$. We modify the decryption oracle with a special rejection rule: if the adversary submits a ciphertext ψ for decryption at a point in time after the encryption oracle has been invoked, such that $(u_1, u_2, e) \neq (u_1^*, u_2^*, e^*)$ but $\alpha = \alpha^*$, then the decryption oracle immediately outputs reject and halts (before executing step **D4′**). to analyze this game, we define two events. first, we define the event C_5 to be the event that the decryption oracle in game $\mathbf{G}_5$ rejects a ciphertext using the special rejection rule. We define the event R_5 to be the event in game $\mathbf{G}_5$ that some ciphertext ψ is submitted to the decryption oracle that is rejected in step **D4′** but that would have passed the test in step **D4**. note that such a ciphertext is not rejected by the special rejection rule, since that rule is applied before step **D4′** is executed. Now, it is clear that games $\mathbf{G}_4$ and $\mathbf{G}_5$ proceed identically until event C_5 occurs. in particular, the events $R_4 \wedge \neg C_5$ and $R_5 \wedge \neg C_5$ are identical. so by difference lemma, we have $|Pr[R_5] - Pr[R_4]| \leq Pr[C_5]$. Now, if event C_5 occurs with non-negligible probability, we immediately get an algorithm that contradicts the target collision resistance assumption;

Lemma 3.4. *There exists a probabilistic algorithm $\mathcal{A}_2$, whose running time is essentially the same as that of $\mathcal{A}$, such that:*

$$Pr[C_5] \leq \mathrm{AdvTCR}_{\mathbf{HF},\mathcal{A}_2}(\lambda|\Gamma) + 1/q. \qquad (3)$$

Finally, we show that event R_5 occurs with negligible probability, based on purely information theoretic considerations:

Lemma 3.5. *We have*

$$Pr[R_5] \leq Q_{\mathcal{A}}(\lambda)/q. \qquad (4)$$

Theorem 3.2. To prove this theorem, let us fix $\mathcal{A}$, λ, and $\Gamma[\hat{G}, G, g_2, q]$. Consider the attack game $\mathbf{G}_0$ as defined above (or see §6.2 in (Cramer and Shoup, 2003)). This is the game that attacker $\mathcal{A}$ plays against the scheme **FCS** for the given values of λ and Γ. We adopt all the notations conventions established at the beginning of §6.2 in (Cramer and Shoup, 2003). We now modify game $\mathbf{G}_0$ to obtain a new game $\mathbf{G}_{\mathbf{FCS}}$.

Game$_{\mathbf{FCS}}$. In this game, we modify the decryption oracle so that in place of steps **D4** and **D5** in **CS1**, we execute steps **D4′** and **D5′** as in the scheme **FCS**. (Note that in the **FCS** encryption algorithm the random parameter generated is r instead of u in **CS1** scheme. So the parameter u^* in **CS1** corresponds to r^* here). We emphasize that in game $\mathbf{G}_{\mathbf{FCS}}$, we have $x = x_1 + x_2 w, y = y_1 + y_2 w$, and $z = z_1 + z_2 w$, where $w, x_1, x_2, y_1, y_2, z_1$, and z_2 are generated by the key generation algorithm of **CS1**.

Let $T_{\mathbf{FCS}}$ be the event that $\sigma = \sigma'$ in game $\mathbf{G}_{\mathbf{FCS}}$. We remind the reader that games $\mathbf{G}_0$ and $\mathbf{G}_{\mathbf{FCS}}$ all operate on the same underlying probability space: all of the variables

$$\mathrm{Coins}, H, w, x_1, x_2, y_1, y_2, z_1, z_2, \sigma, r^*$$

that ultimately determine the events T_0, and $T_{\mathbf{FCS}}$ have the same values in games $\mathbf{G}_0$ and $\mathbf{G}_{\mathbf{FCS}}$; all that changes is the functional behavior of the decryption oracle. It is straightforward to verify that

$$\mathrm{AdvCCA}_{\mathbf{FCS},\mathcal{A}}(\lambda|\Gamma) = |Pr[T_{\mathbf{FCS}}] - 1/2|$$

Let us define the event $R_{\mathbf{FCS}}$ to be the event that some ciphertext is rejected in game $\mathbf{G}_{\mathbf{FCS}}$ in step **D4′** that would have passed the test in **D4** in **CS1** scheme in (Cramer and Shoup, 2003). It is clear that games $\mathbf{G}_0$ and $\mathbf{G}_{\mathbf{FCS}}$ all proceed identically until event $R_{\mathbf{FCS}}$ occurs. In particular, we have the events $T_0 \wedge \neg R_{\mathbf{FCS}}$, and $T_{\mathbf{FCS}} \wedge \neg R_{\mathbf{FCS}}$ are identical. So by difference lemma, we have $|Pr[T_0] - Pr[T_{\mathbf{FCS}}]| \leq Pr[R_{\mathbf{FCS}}]$. So it suffices to show that

$$Pr[R_{\mathbf{FCS}}] \leq Q_{\mathcal{A}}(\lambda)/q. \qquad (5)$$

To do this, for $1 \leq i \leq Q_{\mathcal{A}}(\lambda)$, let $R_{\mathbf{FCS}}^{(i)}$ be the event that there is an i^{th} ciphertext submitted to the decryption oracle in game $\mathbf{G}_{\mathbf{FCS}}$, and that this ciphertext is rejected in step **D4′**, but would have passed the test in step **D4** in **CS1** scheme in (Cramer and Shoup, 2003).

The bound 5 will follow immediately from the following lemma.

Lemma 3.6. *For all $1 \leq i \leq Q_{\mathcal{A}}(\lambda)$, we have $Pr[R_{\mathbf{FCS}}^{(i)}] \leq 1/q$.*

Proof. The proof of this lemma is almost identical to that of Lemma 10 in (Cramer and Shoup, 2003). Note that in game $\mathbf{G}_{\mathbf{FCS}}$, the encryption oracle uses the "real" encryption algorithm, and so itself does not leak any additional information about (x_1, x_2, y_1, y_2). This is in contrast to game $\mathbf{G}_5$, where the encryption oracle does leak additional information.

Fix $1 \leq i \leq Q_{\mathcal{A}}(\lambda)$. Consider the quantities:

$$X := (\mathrm{Coins}, H, w, z, \sigma, r^*) \text{ and } X' := (x, y).$$

The values of X and X' completely determine the adversary's entire behavior in game $\mathbf{G}_5$, and hence determine if there is an i^{th} decryption oracle query, and if so, the value of the corresponding ciphertext. Let us call X and X' relevant if for these values of X and X', there is an i^{th} decryption oracle query, and the corresponding ciphertext passes steps **D1** and **D2** in **CS1** scheme in (Cramer and Shoup, 2003).

It will suffice to prove that conditioned on any fixed, relevant values of X and X', the probability that $R_{\mathbf{FCS}}^{(i)}$ occurs is bounded by $1/q$.

The remainder of the argument is exactly as in Lemma 10 in (Cramer and Shoup, 2003), except using X, X', and the notion of relevant as defined here. □

□

4 Performances Evaluation

We study the complexity and the performance of Fast Cramer-Shoup encryption scheme with the following variants of Cramer-Shoup encryption scheme: *Original CS* protocol described in Section 2.1 and represented by blue squares in all the figures, *Standard CS* protocol presented in Section 2.2 and represented by red asterisk points in all the curves, *Efficient CS* protocol given in Section 2.3 represented by green triangles in all the plots, *Fast Cramer-Shoup* protocol is introduced in Section 3 and represented by black circles in all the figures.

We study the number of operations performed in each algorithm: key generation, encryption and decryption. We present complexity results in Table 1 to Table 4. This complexity study is confirmed by an experimental comparison where all the presented algorithms have been implemented with the library GMP (Granlund and the GMP development team, 2020).

The tests are performed with security parameters of size 512, 1024, 2048 and 4096 bits. The average execution time of all schemes is carried out under the same conditions in terms of generation of the values and sizes of the security parameters. Indeed, given the size of a security parameter, we perform 1000 trials with new parameters: a prime number and some random plaintexts that are generated for each trial. We use those parameters for the performance evaluation of the four protocols. Then, we measure the execution time during one execution of each algorithm. Finally, we compute the mean value for each algorithm[1]. The key generation algorithms are compared in Figure 1, the encryption algorithms are compared in Figure 2 and decryption algorithms are compared in Figure 3. Note that the study of the decryption algorithm is more detailed, as we have divided it into two phases. The first step is the verification phase (that checks integrity of the received message) and the second is the actual decryption (where the plaintext is retrieved). The decryption comparison holds three charts, one for the verification (Figure 4), another for the decryption itself (Figure 5) and a third one for the full decryption algorithm (verification and decryption phases, see Figure 3).

[1]We do not present the corresponding standard deviations since they are very low.

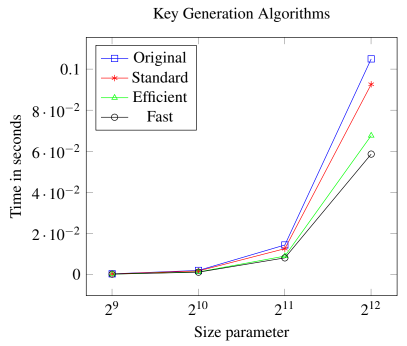

Figure 1: Average execution time for key generation algorithms depending of the security parameter size.

Key Generation				
Number of	Original	Standard	Efficient	Fast
Generator	2	2	**1**	**1**
Random	6	5	**4**	**4**
Multiplication	3	2	**0**	3
Exponentiation	6	5	**4**	**4**

Table 1: Comparison of Original, Standard, Efficient and Fast Cramer-Shoup (minimum in bold).

We also analyze these performance evaluations between the four algorithms (key generation, encryption and decryption) in terms of their theoretical complexity.

4.1 Key Generation Algorithms

By studying the complexity of the key generation algorithm of Table 1, we notice that protocol *Fast CS* and *Efficient CS* are the fastest in terms of number of operations. The difference lies in the multiplication where *Fast CS* has three of them while *Efficient CS* has none. One would expect a longer execution time for *Fast CS*. Nonetheless, we observe in Figure 1 that their execution time is similar.

It can be explained by the fact that the *Fast CS* key generation algorithm computes an Euclidean division leading to $kq = wt + z$ where k is generated in $\mathbb{Z}_q$. Hence its size is the same as q but t is generated in $\mathbb{Z}^*_{\sqrt{q}}$ and its size is half of q. Since z is the remainder of the euclidean division, its order size is the same as [2] t. Thus the cost for computing $h = g^z$ is half of the cost for computing $h = g^z$ where z has an order size equal to the size of q. This gain seems to be compensated with the three additional multiplications leading to similar execution for the overall key generation algorithm between *Fast CS* and *Efficient CS* protocols.

[2]We recall that the rest of the euclidean division is always less than the quotient, i.e., $0 \leq z < w$

Public Key Parameters				
Number of	Original	Standard	Efficient	Fast
Elements	**5**	**5**	**5**	**5**
Secret Key Parameters				
Elements	6	5	**4**	**4**

Table 2: Comparison of Original, Standard, Efficient and Fast Cramer-Shoup for key parameters.

To conclude for key generation, the two fastest algorithms are *Efficient CS* and *Fast CS*, while the two slowest are *Original CS* and *Standard CS*. As the complexity analysis shows *Original CS* to be naturally slower than *Standard CS* since it has one more exponentiation, one more multiplication and one more random numbers than *Original CS*. Moreover, the number of secret parameters are also in favour of *Efficient CS* and *Fast CS* as depicted in Table 2. We manage to reduce this number to four elements while *Original CS* and *Standard CS* have six and five respectively.

4.2 Encryption Algorithms

In Table 3, we compare for each step *Original*, *Standard*, *Efficient* and *Fast* protocols. We can see that all schemes have the same number of exponents, multiplication and generation of random number in the encryption algorithm. Thus the average execution time is expected to be the same.

Encryption				
Number of	Original	Standard	Efficient	Fast
Random	**1**	**1**	**1**	**1**
Multiplication	**3**	**3**	**3**	**3**
Exponentiation	**5**	**5**	**5**	**5**

Table 3: Comparison of Original, Standard, Efficient and Fast Cramer-Shoup for encryption.

The curves of Figure 2 give the average execution time of encryption algorithms. As seen in the complexity analysis, we observe empirically that timings are equal for all protocols since the number of computations are equals. In all protocols, the ciphertexts are mainly computed with parameters of the same order size. Thus the average execution time is the same in all cases.

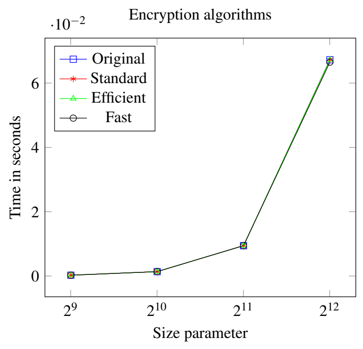

Figure 2: Average execution time for encryption algorithms.

4.3 Decryption Algorithms

The curves of Figure 3 represent the average execution time of decryption algorithms. We observe that the fastest one is Fast Cramer-Shoup then we find the *Efficient CS* and naturally *Standard CS* follows by *Original CS*. The fact that Fast Cramer-Shoup is faster that *Efficient CS* is not obvious when we are looking at the number of total computations. To explain this we need to have a more fine analysis of these two decryption algorithms.

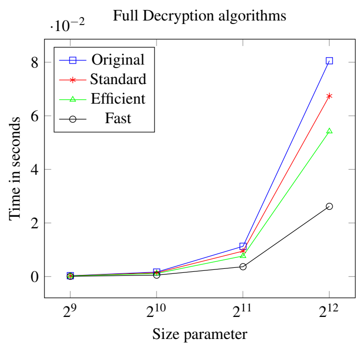

Figure 3: Average execution time for decryption algorithms.

There are two components in this algorithm, the verification phase and the actual decryption. The verification is used for checking the integrity of the message if the outputs are valid (meaning that the message has not been modified by an unexpected party) and the decryption computation is executed leading to retrieve the plaintext. In Figure 3, we observe that Fast Cramer-Shoup has a faster execution time. We give further details to study and understand such improvement. We give two charts, one for verification phase in Figure 4 and one for the actual decryption in Figure 5.

Verification phase. We observe in Figure 4 that the average execution time of the *Original CS* and *Standard CS* protocols is the same. This is expected since the verification step is the same in both cases. The *Fast CS* and *Efficient CS* protocols have the same average execution time. In both cases the second verification is the same: $u_1^{x+\alpha y} = v$. The difference is then on the first verification. The *Efficient CS* protocol uses one modular exponent, $u_1^w = u_2$; while ours uses two modular exponents, $\beta u_2^z = u_1^w u_2^z = 1$. Despite the result given in Table 4, we need to perform

		Verification		
Number of	Original	Standard	Efficient	Fast
Sum	2	2	**1**	**1**
Multiplication	3	3	**1**	2
Exponentiation	**2**	**2**	**2**	3

		Decryption		
Number of	Original	Standard	Efficient	Fast
Inverse	2	1	1	**0**
Multiplication	2	**1**	**1**	**1**
Exponentiation	2	1	1	**0**

Table 4: Comparison for decryption algorithms.

a more precise analysis. Indeed, there is actually no difference between those two computations in terms of execution time since elements w and z of *Fast CS* have their size equal to half the size of w in the *Efficient CS*. Hence, the overall execution time for the verification of our protocol *Fast CS* is the same as *Efficient CS* protocol.

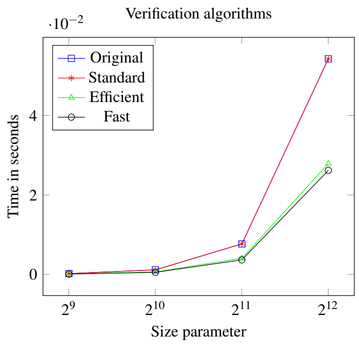

Figure 4: Average execution time for verification phases.

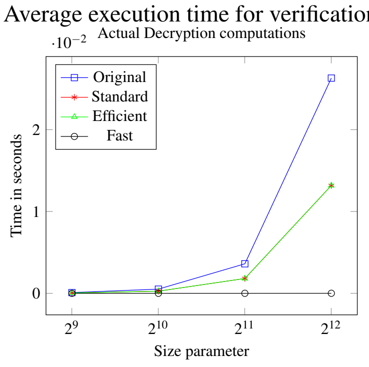

Figure 5: Average execution time for decryption phases.

The Decryption phase. It is clear that *Fast CS* protocol is faster than *Original* for decryption since the number of exponents is reduced. We can also say from Table 4 that *Fast CS* is faster than *Standard CS* and *Efficient CS* at decryption.

We observe in Figure 5 that *Standard CS* and *Efficient CS* have the same execution time. This result is expected since they both use the same computation with the same elements size, i.e., $eu_1^{-z} = m$. The latter computation uses one exponentiation and one inversion to end with a multiplication.

The Fast Cramer-Shoup decryption algorithm requires to recover m as follows $\beta e = m$ where β has been computed in the verification phase. Thus, the decryption consists of only one multiplication. The gain comes from the fact that we have one inverse less to compute, and also one exponentiation less to compute. The gain of the overall execution time of decryption phase for our protocol is more than 60%. Note that the curve related to Fast Cramer-Shoup is not zero but since this curve is composed of only one multiplication, the comparison with a modular exponent, an inversion and a multiplication, is in favor of the stand alone multiplication.

5 Conclusion

We propose an IND-CCA2 Public Key cryptosystem called Fast Cramer-Shoup. It is an improvement of Cramer-Shoup encryption scheme that is faster. We prove the IND-CCA2 security of this new scheme. We also implement in GMP (Granlund and the GMP development team, 2020) our scheme to compare it to Cramer-Shoup schemes. In the future, we aim at applying our technique to other schemes that are based on Cramer-Shoup like for instance (Kurosawa and Trieu Phong, 2014; Abdalla et al., 2015).

REFERENCES

Abdalla, M., Benhamouda, F., and Pointcheval, D. (2015). Public-key encryption indistinguishable under plaintext-checkable attacks. In Katz, J., editor, *Public-Key Cryptography – PKC 2015*, pages 332–352, Berlin, Heidelberg. Springer Berlin Heidelberg.

Barthe, G., Grégoire, B., Lakhnech, Y., and Zanella Béguelin, S. (2011). Beyond provable security. Verifiable IND-CCA security of OAEP. In *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196. Springer.

Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS 93, page 6273, New York, NY, USA. Association for Computing Machinery.

Bellare, M. and Rogaway, P. (1994). Optimal asymmetric encryption. In Santis, A. D., editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer.

Boneh, D. (1998). The decision diffie-hellman problem. In *In Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423, pages 48–63.

Cramer, R. and Shoup, V. (1998a). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proc. of the 18th Annual International Cryptology Conference on Advances in Cryptology, crypto'98*, pages 13–25.

Cramer, R. and Shoup, V. (1998b). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive: Report 1998/006.

Cramer, R. and Shoup, V. (2003). Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226.

Cramer, R. and Shoup, V. (May 2002). Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *In L. Knudsen, editor, Proceedings of Eurocrypt 2002*, volume 2332 of LNCS, pages 45–64.

Dolev, D., Dwork, C., and Naor, M. (1991). Non-malleable cryptography. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC 91, page 542552, New York, NY, USA. Association for Computing Machinery.

Elgamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO, IT-31(4)*, volume 4, pages 469–472.

Fujisaki, E. and Okamoto, T. (1999). How to enhance the security of public-key encryption at minimum cost. In *Public Key Cryptography*, pages 53–68, Berlin, Heidelberg. Springer Berlin Heidelberg.

Granlund, T. and the GMP development team (2020). *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.2.0 edition.

International Organization for Standardization, Genève, S. (2006). Iso/iec 18033-2:2006, information technology security techniques encryption algorithms part 2: Asymmetric ciphers. https://www.shoup.net/iso/std4.pdf.

Joux, A. and Guyen, K. (2006). Separating decision diffie-hellman to diffie-hellman in cryptographic groups.

Kurosawa, K. and Trieu Phong, L. (2014). Kurosawa-desmedt key encapsulation mechanism, revisited. In Pointcheval, D. and Vergnaud, D., editors, *Progress in Cryptology – AFRICACRYPT 2014*, pages 51–68, Cham. Springer International Publishing.

Naor, M. and Yung, M. (1989). Universal one-way hash functions and their cryptographic applications. In *In 21st Annual ACM Symposium on Theory of Computing*.

Paillier, P. (May 1999). Public-key cryptosystems based on composite degree residuosity classes. In *In J. Stern, editor, Proceedings of Eurocrypt 1999*, volume 1592 of LNCS, pages 223–38.

Phan, D. H. and Pointcheval, D. (2004). Oaep 3-round: A generic and secure asymmetric encryption padding. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 63–77. Springer.

Pointcheval, D. (2011). *OAEP: Optimal Asymmetric Encryption Padding*, pages 882–884. Springer US, Boston, MA.

Shoup, V. (2001). Oaep reconsidered. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO 01, page 239259, Berlin, Heidelberg. Springer-Verlag.

Shoup, V. and Gennaro, R. (1998). Securing threshold cryptosystems against chosen ciphertext attack. In *In Advances in Cryptology-Eurocrypt '98*.

Sow, D. and Sow, D. (2011). A new variant of el gamal's encryption and signatures schemes. *JP Journal of Algebra, Number Theory and Applications*, 20(1):21–39.