Self Evaluation of PSEC-KEM

1 Introduction

This document describes the security assessment and performance on PSEC-KEM, which is a key distribution scheme in the public-key cryptosystem.

2 Design Policy and the Underlying Theory and Techniques

PSEC-KEM is a version (PSEC-KEM in [4]) modified to KEM (Key Encapsulation Mechanism) from a public-key encryption, PSEC-2, which is a version [1] with the strongest security converted from the primitive elliptic curve encryption function, elliptic curve ElGamal encryption function.

PSEC-KEM inherits the various practical merits of the elliptic curve encryption function (elliptic curve ElGamal encryption function), and is proven to has the strongest level of security (semantic security against adaptive chosen-ciphertext attacks: IND-CCA2) under some assumptions.

We will show its security and performance results below.

2.1 Summary of Security

PSEC-KEM is semantically secure against adaptive chosen-ciphertext attacks (IND-CCA2) (or non-malleable against adaptive chosen-ciphertext attacks (NM-CCA2)) in the random oracle model, under the elliptic curve computational Diffie-Hellman (CDH) assumption.

We will compare the security of PSEC-KEM with other encryption schemes such as the elliptic curve (EC-)Cramer-Shoup, ECIES and elliptic curve (EC-)ElGamal. The following table summarizes the comparison of security.

Table 1: Comparison of Security

Scheme	Provably Secure? (IND-CCA2?)	Number-theoretical assumption	Functional assumption	Reduction efficiency
PSEC-KEM(OTP)	Yes	EC-CDH	Truly random	**
EC-Cramer-Shoup	Yes	EC-DDH	UOWHF	*
ECIES	Yes	EC-GDH	Truly random	**
EC-ElGamal	No (broken)	–	–	–

EC-CDH, EC-DDH and EC-GDH denote the elliptic curve versions of computational Diffie-Hellman, decisional Diffie-Hellman and gap Diffie-Hellman assumptions, respectively. ** denotes that the efficiency is almost optimal, and * denotes that the efficiency is less than the almost optimal cases.

2.2 Summary of Efficiency

Here we compares the efficiency of PSEC-KEM with those of other encryption schemes, EC-Cramer-Shoup, ECIES, and EC-ElGamal.

To compare the schemes under the equal conditions, we assume that the plaintext size is 128 bits for all schemes, since public-key encryption schemes are usually employed for key distribution of a symmetric-key encryption (128 key is the most typical key size of symmetric-key encryptions).

The parameter sizes for EL-Cramer-Shoup, ECIES, EC-ElGamal are assumed to be the same as those of PSEC-KEM. In this table, we show the required number of elliptic curve multiplications for each operation.

Table 2: Comparison of Efficiency

Scheme	Encryption	Decryption
PSEC-KEM	2	2
EC-Cramer-Shoup	5	3 (4*)
ECIES	2	1 (2*)
EC-ElGamal	2	1

$*$: The case when an additional multiplication is required to check whether a ciphertext is in the subgroup generated by the base point.

3 Security Proof of PSEC-KEM

In this section, we will prove that PSEC-KEM is semantically secure against adaptively chosen-ciphertext attacks, under the the elliptic curve computational Diffie-Hellman assumption and in the random oracle model.

Here, we view MGF as a random oracle. This effectively gives us two independent random oracles,

$$G : \mathbf{B}_{hLen} \to \mathbf{B}_{pLen+128+KeyLen},$$

$$H : \mathbf{B}_{32+2\cdot qmLen} \to \mathbf{B}_{hLen}.$$

For simplicity, we denote EC for ECP2OSP$(C_1, qmLen)$ and EQ for ECP2OSP$(Q, qmLen)$.

Theorem 3.1 *Let $\mathcal{A}$ be an (adaptively chosen-ciphertext attack) CCA2–adversary against the "semantic security" (IND) of PSEC-KEM $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, with advantage ε and running time t, making q_D, q_G and q_H queries to the decryption oracle, and the random oracles G and H respectively. Then, there exists an algorithm with the success probability, ε', of providing a list of $(q_H + q_D)$ strings which* *includes a correct answer of the elliptic curve computational Diffie-Hellman (CDH) problem regarding $\mathcal{K}$ and the running time t', such that*

$$\varepsilon' \geq \frac{\varepsilon}{2(1+2^{-128})} - \frac{(q_G + 3q_D)(1+2^{-128})}{p} - \frac{2q_D + q_G}{2^{hLen}},$$

and

$$t' \leq t + q_H \cdot (T + \mathcal{O}(1)),$$

where T denotes the time complexity of computing two elliptic curve multiplication operations regarding $\mathcal{K}$.

Note: If there exists an algorithm which outputs a list of $(q_H + q_D)$ strings including a correct answer of the computational Diffie-Hellman (CDH) problem, then we can efficiently obtain the correct answer of the computational Diffie-Hellman (CDH) problem, using the algorithm (by the random self-reducible property of the CDH problem) [3, 4].

Hereafter, we will repeatedly use the following simple result:

Lemma 3.2 *For any probability events* E, F *and* G

$$\Pr[\mathsf{E} \wedge \mathsf{F} \mid \mathsf{G}] \leq \begin{cases} \Pr[\mathsf{E} \mid \mathsf{F} \wedge \mathsf{G}] \\ \Pr[\mathsf{F} \mid \mathsf{G}]. \end{cases}$$

We prove theorem 3.1 in three stages. The first presents the reduction of algorithm $\mathcal{B}$ for breaking the Computational Diffie-Hellman (CDH) problem to the IND-CCA2 adversary $\mathcal{A}$ for PSEC-KEM. The second shows that the decryption oracle simulation employed in this reduction works correctly with overwhelming probability. Finally, we analyze the success probability of our reduction in total, through the incorporation of the above-mentioned analysis of the decryption oracle simulation.

Note: The definition of IND(i.e., semantic security) in the scenario of the key encapsulation mechanism such as PSEC-KEM is a bit different from the (public-key) encryption. See Section 2.2 in [4] for the definition.

3.1 Description of the Reduction

In this first part, we recall how reduction operates. Let $\mathcal{A}$ be an adversary against the semantic security of PSEC-KEM with $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, under chosen-ciphertext attacks. Within time bound t, $\mathcal{A}$ asks q_D, q_G and q_H queries to the decryption oracle and the random oracles G and H respectively, and distinguishes key K (either the correct key or just a random string) with an advantage greater than ε. Let us describe the reduction $\mathcal{B}$.

3.1.1 Top Level Description of the Reduction.

1. $\mathcal{B}$ is given public key PK including two point P and W on E and another point C_1^*. The aim of $\mathcal{B}$ is to obtain a list of data including the DH solution, Q^*, for (P, W, C_1^*) such that $\log_P W = \log_{C_1^*} Q^*$ $(= s)$.

2. $\mathcal{B}$ randomly selects a bit b and two random strings, $c_2^* \in \mathbf{B}_{hLen}$ and $K \in \mathbf{B}_{keyLen}$. $\mathcal{B}$ runs A on the public data, ciphertext $c^* = (EC^*, c_2^*)$ with $EC^* = \text{ECP2OSP}(C_1^*, qmLen)$ and K. $\mathcal{B}$ simulates the answers to the queries of A to the decryption oracle and random oracles G and H respectively. See the description of these simulations below.

3. A finally outputs answer b'. $\mathcal{B}$ then outputs the list of queries asked to H (especially its EQ part), in which EQ^* (i.e., Q^*) may be included.

3.1.2 Simulation of Random Oracles G and H.

The random oracle simulation has to simulate the random oracle answers, managing query/answer lists G-List and H-List for the oracles G and H respectively, both are initially set to empty lists:

- For a fresh query $\delta = (EC, EQ)$ with $EC \neq EC^*$ to H, the simulator outputs a random value $H(\delta)$, and the pair $(\delta, H(\delta))$ is concatenated to the H-List. For a fresh query r to G, it outputs a random value $G(r)$, and the pair $(r, G(r))$ is concatenated to the G-List.

- For a fresh query $\delta^* = (EC^*, EQ)$ to H, the simulator randomly selects $H(\delta^*)$ and the pair $(\delta^*, H(\delta^*))$ is concatenated to H-List. It calculates $r = c_2^* \oplus H(\delta^*)$, and looks for a query r in the $GList$. When it does not exist, the simulator randomly selects $t \in \mathbf{B}_{pLen+128}$ as $t = [G(r)]^{pLen+128}$, i.e., the $(pLen+128)$ bit prefix of $G(r)$. Then, it calculates $\alpha = \text{OS2IP}(t) \bmod p$, and checks whether $C_1^* = \alpha P$. If it holds, it calculates $Q^* = \alpha W$, which is the solution of the DH problem with (P, W, C_1^*) (and outputs Q^*). If $EQ = \text{ECP2OSP}(Q^*, qmLen)$, it sets $[G(r)]_{keyLen} = K$ if $b = 0$, and it randomly selects $[G(r)]_{keyLen}$ if $b = 1$. In this case, (EC^*, c_2^*) is a valid output (or correctly simulated output) of the encryption oracle in the real scenario of IND-CCA2. If $C_1^* \neq \alpha P$, it randomly selects $[G(r)]_{keyLen}$.

3.1.3 Simulation of the Decryption Oracle.

On query $c = (EC, c_2)$ to the decryption oracle, decryption oracle simulation $\mathcal{DS}$ looks at query-answer $(\delta, H(\delta)) \in$ H-List such that $\delta = (EC, EQ)$ (for any EQ). If no such pair is found in H-List, "Reject" is returned. If such pair exists, it calculates $r = c_2 \oplus H(\delta)$, and it looks for a query r in the $GList$. If it does not exist, it randomly selects G's answer to r, $G(r)$, and the pair $(r, G(r))$ is concatenated to the G-List. Then, it calculates $\alpha = \text{OS2IP}(t) \bmod p$, where $t = [G(r)]^{pLen+128}$. It checks whether $EC = \text{ECP2OSP}(\alpha P)$ and $EQ = \text{ECP2OSP}(\alpha W)$. If either one of them does not hold, "Reject" is returned. If the both equations hold, $\mathcal{DS}$ outputs $[G(r)]_{keyLen}$, i.e., $keyLen$ bit suffix of $G(r)$, as key k.

3.1.4 Remarks.

When we have found Q^*, we could output the expected result Q^* and stop the reduction. But for this analysis, we assume the reduction goes on and that $\mathcal{B}$ only outputs it, or the list of queries asked to H, once A has answered b' (or after a time limit).

The distribution of α in the simulation is a bit different from that in the real situation: in the simulation α is distributed uniformly in $\{0, 1, \ldots, p-1\}$, but in the real situation it is a bit biased as that $\alpha = \mathrm{OS2IP}(t) \bmod p$ and t is uniform in $\mathbf{B}_{pLen+128}$.

Since our coding method of an elliptic curve point is a one-to-one and onto mapping, $C_1 = C_1'$ if and only if $EC = EC'$, and $Q = Q'$ if and only if $EQ = EQ'$.

3.2 Notations

In order to proceed to the analysis of the success probability of the above-mentioned reduction, one needs to set up notations. First, we still denote with a star (*) all variables related to the challenge ciphertext $c^* = (EC^*, c_2^*)$, obtained from the encryption oracle. All other variables refer to the decryption query c, asked by the adversary to the decryption oracle, and thus to be decrypted by this simulation. We consider several events about queries to the random oracles and the decryption oracle:

- AskH denotes the event that query (EC^*, EQ^*) has been asked to H, and AskG denotes the event that query r^* has been asked to G.
- GBad is the event that r^* $(= c_2^* \oplus H(EC^*, EQ^*))$ is asked to G oracle and $EC^* \neq \mathrm{ECP2OSP}(\alpha P)$, or $EC^* = \mathrm{ECP2OSP}(\alpha P)$ but $[G(r^*)]_{keyLen} \neq K$ if $b = 0$, where $\alpha = \mathrm{OS2IP}(t^*) \bmod p$ and $t^* = [G(r^*)]^{pLen+128}$ (bit b is fixed in the reduction scenario). Note that the event GBad implies AskG. As seen above, GBad is the only event that makes the simulation imperfect, in the chosen-plaintext attack scenario.
- Fail denotes the event that the above decryption oracle simulator outputs at least one wrong decryption answer among q_D answers.
- $\mathsf{Bad} = \mathsf{GBad} \vee \mathsf{Fail}$.
- CBad denotes the union of the bad events, $\mathsf{CBad} = \mathsf{RBad} \vee \mathsf{EBad}$, where
  - EBad denotes the event that $EC = EC^*$;
  - RBad denotes the event that $r = r^*$;
- AskRE denotes the intersection of both events about the oracle queries, $\mathsf{AskRE} = \mathsf{AskR} \wedge \mathsf{AskE}$, where
  - AskR denotes the event that r $(= c_2 \oplus H(EC, EQ))$ has been asked to G;

– AskE denotes the event that (EC, EQ) has been asked to H;

Note that the Fail event is limited to the situation in which $\mathcal{DS}$ rejects a ciphertext whereas it would be accepted by the actual decryption oracle. Indeed, when $\mathcal{DS}$ accepts, we see that the ciphertext is actually valid and corresponds to the output plaintext.

3.3 Analysis of the Decryption Oracle Simulation

We analyze the success probability of decryption oracle simulator $\mathcal{DS}$.

3.3.1 Security Claim.

Lemma 3.3 *When at most one ciphertext $c^* = (EC^*, c_2^*)$ has been directly obtained from the encryption oracle, the decryption oracle simulation $\mathcal{DS}$ can correctly produce the decryption oracle's answers on q_D queries (ciphertext; $c = (EC, c_2)$, $c \neq c^*$) with probability greater than ε_1, within time bound t_1, where*

$$\varepsilon_1 \geq 1 - \left(\frac{(q_G + 3q_D)(1 + 2^{-128})}{p} + \frac{q_D}{2^{hLen}} \right),$$

$$t_1 \leq q_H \cdot (T + \mathcal{O}(1)),$$

and T is the computational complexity of the operations of αP and αW.

Before we start the analysis, we recall that the decryption oracle simulator is given the ciphertext c to be decrypted, as well as the ciphertext c^* obtained from the encryption oracle and both the G-List and H-List resulting from the interactions with the simulator of the random oracles G and H. If the ciphertext has been correctly built by the adversary (r has been asked to G and (EC, EQ) to H), the simulation will output the correct answer. However, it will output "Reject" in any other situation, whereas the adversary may have built a valid ciphertext without asking both queries to the random oracles G and H.

3.3.2 Success Probability.

Since our goal is to show the probability of the event Fail. Granted $\neg$CBad $\wedge$ AskRE, the simulation is perfect, and cannot fail. Thus, we have to consider the complementary events:

$$\Pr[\mathsf{Fail}] = \Pr[\mathsf{Fail} \wedge \mathsf{CBad}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskRE}].$$

First we focus on the former term, $\Pr[\mathsf{Fail} \wedge \mathsf{CBad}]$. CBad = RBad $\vee$ EBad, but RBad never occurs. This is because: $r = r^*$ implies that $\alpha = \alpha^*$, $C_1 = C_1^*$ (i.e., $EC = EC^*$), $Q = Q^*$, and $H(EC, EQ) = H(EC^*, EQ^*)$. Hence $c_2 = c_2^*$, i.e., $c = (EC, c_2)$ is equivalent to $c^* = (EC^*, c_2^*)$. Such c is not allowed as a query to the decryption oracle. Thus, CBad = EBad. We will evaluate $\Pr[\mathsf{Fail} \wedge \mathsf{EBad}]$, which is the probability that there exists at least one c among q_D queries to the decryption oracle such that $c = (EC^*, c_2)$ with $c_2 \neq c_2^*$ (i.e., $r \neq r^*$) satisfies $\mathrm{OS2IP}([G(r)]^{pLen+128}) \equiv \mathrm{OS2IP}([G(r^*)]^{pLen+128}) \pmod p$. The probability to satisfy the equation for each c is at most $\frac{1+2^{-128}}{p}$. The adversary has a (potential) chance to ask q_G queries to check whether the equation holds, and an additional chance for q_D queries to decryption oracle. So in total,

$$\Pr[\mathsf{Fail} \wedge \mathsf{EBad}] \leq \frac{(q_G + q_D)(1 + 2^{-128})}{p}.$$

We now evaluate the latter term, $\Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskRE}]$. Since $\mathsf{CBad} = EBad$ and $\mathsf{AskRE} = \mathsf{AskR} \wedge \mathsf{AskE}$, it is

$$\Pr[\mathsf{Fail} \wedge \neg\mathsf{EBad} \wedge (\neg\mathsf{AskR} \vee \neg\mathsf{AskE})]$$

$$\leq \Pr[\mathsf{Fail} \wedge \neg\mathsf{EBad} \wedge \neg\mathsf{AskR}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{EBad} \wedge \mathsf{AskR} \wedge \neg\mathsf{AskE}].$$

For the former term, since r is never asked to G and independent from r^*, the probability that c is valid (i.e., $\alpha = \mathrm{OS2IP}([G(r)]^{pLen+128}) \pmod p$ and $EC = \mathrm{ECP2OSP}(\alpha P)$) is at most $\frac{1+2^{-128}}{p}$. Since the adversary has a chance for q_D queries to decryption oracle,

$$\Pr[\mathsf{Fail} \wedge \neg\mathsf{EBad} \wedge \neg\mathsf{AskR}] \leq \frac{q_D(1 + 2^{-128})}{p}.$$

For the latter term, $\Pr[\mathsf{Fail} \wedge \neg\mathsf{EBad} \wedge \mathsf{AskR} \wedge \neg\mathsf{AskE}]$, r is queried to G, but (EC, EQ) is never asked to H and independent from (EC^*, EQ^*). So the probability that $r' = c_2 \oplus H(EC, EQ)$ is equivalent to r or $G(r')$ happens to be consistent with C_1 is at most $\frac{1}{2^{hLen}} + \frac{1+2^{-128}}{p}$. Since the adversary has a chance for q_D queries to the decryption oracle,

$$\Pr[\mathsf{Fail} \wedge \neg\mathsf{EBad} \wedge \neg\mathsf{AskR}] \leq q_D(\frac{1}{2^{hLen}} + \frac{1 + 2^{-128}}{p}).$$

As a consequence,

$$\Pr[\mathsf{Fail}] \leq \frac{(q_G + 3q_D)(1 + 2^{-128})}{p} + \frac{q_D}{2^{hLen}}.$$

The running time of this simulator includes just the computation time of αP and αW for all α values obtained from ciphertext $c = (EC, c_2)$, the corresponding queries, (EC, EQ), in $\mathsf{H\text{-}List}$, and the corresponding r values of $\mathsf{G\text{-}List}$ and is thus at most

$$q_H \cdot (T + \mathcal{O}(1)),$$

where T is the computational complexity of the operations of αP and αW.

3.4 Success Probability of the Reduction

This subsection analyzes the success probability of our reduction with respect to the advantage of the IND-CCA2 adversary of PSEC-KEM. The goal of the reduction is, given the elliptic curve parameters and (P, W, C_1^*), to obtain a list of q_H values including Q^*. Therefore, the success probability is obtained by the probability that event AskH occurs during the reduction.

We thus evaluate $\Pr[\mathsf{AskH}]$ by splitting event AskH according to event Bad.

$$\Pr[\mathsf{AskH}] = \Pr[\mathsf{AskH} \wedge \mathsf{Bad}] + \Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}].$$

First let us evaluate the first term.

$$\begin{aligned}
\Pr[\mathsf{AskH} \wedge \mathsf{Bad}] &= \Pr[\mathsf{Bad}] - \Pr[\neg\mathsf{AskH} \wedge \mathsf{Bad}] \\
&\geq \Pr[\mathsf{Bad}] - \Pr[\neg\mathsf{AskH} \wedge \mathsf{GBad}] - \Pr[\neg\mathsf{AskH} \wedge \mathsf{Fail}] \\
&\geq \Pr[\mathsf{Bad}] - \Pr[\mathsf{GBad} \,|\, \neg\mathsf{AskH}] - \Pr[\mathsf{Fail}] \\
&\geq \Pr[\mathsf{Bad}] - \Pr[\mathsf{AskG} \,|\, \neg\mathsf{AskH}] - \Pr[\mathsf{Fail}] \\
&\geq \Pr[\mathsf{Bad}] - \frac{(q_G + 3q_D)(1 + 2^{-128})}{p} - \frac{2q_D + q_G}{2^{hLen}}.
\end{aligned}$$

Here, $\Pr[\mathsf{Fail}] \leq \frac{(q_G+3q_D)(1+2^{-128})}{p} + \frac{q_D}{2^{hLen}}$ is directly obtained from lemma 3.3, and $\Pr[\mathsf{GBad} \,|\, \neg\mathsf{AskH}] \leq \Pr[\mathsf{AskG} \,|\, \neg\mathsf{AskH}]$ is obtained from the fact that event GBad implies AskG. When $\neg\mathsf{AskH}$ occurs, $H(EC^*, EQ^*)$ is unpredictable, and $r^* = c_2^* \oplus H(EC^*, EQ^*)$ is also unpredictable. Hence $\Pr[\mathsf{AskG} \,|\, \neg\mathsf{AskH}] \leq \frac{q_G+q_D}{2^{hLen}}$.

We then evaluate the second term.

$$\begin{aligned}
\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] &= \Pr[\neg\mathsf{Bad}] \cdot \Pr[\mathsf{AskH} \,|\, \neg\mathsf{Bad}] \\
&\geq \Pr[\neg\mathsf{Bad}] \cdot \Pr[\mathcal{A} = b \wedge \mathsf{AskH} \,|\, \neg\mathsf{Bad}] \\
&\geq \Pr[\neg\mathsf{Bad}] \cdot (\Pr[\mathcal{A} = b \,|\, \neg\mathsf{Bad}] - \Pr[\mathcal{A} = b \wedge \neg\mathsf{AskH} \,|\, \neg\mathsf{Bad}]) .
\end{aligned}$$

Here, when $\neg\mathsf{AskH}$ occurs, $H(EC^*, EQ^*)$ is unpredictable, thus $r^* = c_2^* \oplus H(EC^*, EQ^*)$ is unpredictable, and so is b as well. This fact is independent from event $\neg\mathsf{Bad}$. Hence $\Pr[\mathcal{A} = b \wedge \neg\mathsf{AskH} \,|\, \neg\mathsf{Bad}] \leq \Pr[\mathcal{A} = b \,|\, \neg\mathsf{AskH} \wedge \neg\mathsf{Bad}] = 1/2$. Since the distribution of $c^* = (EC^*, c_2^*)$ in this simulation scenario is a bit different ($(1 + 2^{-128})$ times biased) from that of c^* in the real situation,

$$\frac{\varepsilon}{2(1 + 2^{-128})} + \frac{1}{2} \leq \Pr[\mathcal{A} = b] \leq \Pr[\mathcal{A} = b \,|\, \neg\mathsf{Bad}] \cdot \Pr[\neg\mathsf{Bad}] + \Pr[\mathsf{Bad}].$$

Therefore,

$$\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] \geq \left(\frac{\varepsilon}{2(1 + 2^{-128})} + \frac{1}{2} - \Pr[\mathsf{Bad}]\right) - \frac{\Pr[\neg\mathsf{Bad}]}{2} = \frac{\varepsilon/(1 + 2^{-128}) - \Pr[\mathsf{Bad}]}{2}.$$

Combining the evaluation for the first and second terms, and from the fact that $\Pr[\mathsf{Bad}] \geq 0$, one gets

$$\Pr[\mathsf{AskH}] \geq \frac{\varepsilon}{2(1 + 2^{-128})} - \frac{(q_G + 3q_D)(1 + 2^{-128})}{p} - \frac{q_D + q_G}{2^{hLen}}.$$

3.5 Complexity Analysis.

Since the major part of this reduction complexity is for the simulation of the decryption oracle, the time complexity of the overall reduction is

$$t' = t + q_H \cdot (T + \mathcal{O}(1)),$$

where T is the computational complexity of the operations of αP and αW.

4 Evaluation by Implementation

In this section, we show the speed and memory usage when we implement ESIGN by C language. We use the recommended parameters in our specification.

The environment is followings:

CPU	Intel Pentium-III 600MHz
Memory	128 Mbytes
OS	Microsoft Windows2000 SP2
Compiler	Microsoft Visual Studio 6.0 Enterprise Edition

We measure the speed by counting the CPU clock. We show the average speed of 1000 random data test.

The speed is following:

Key generation	5.64 ms
Encryption	11.09 ms
Decryption	10.97 ms

The memory usage is following:

Key generation	7.30 Kbytes
Encryption	2.64 Kbytes
Decryption	2.39 Kbytes

References

[1] Fujisaki, E. and Okamoto, T.: Secure Integration of Asymmetric and Symmetric Encryption Schemes, Proc. of Crypto'99, Springer-Verlag, LNCS 1666, pp. 535–554 (1999).

[2] Bailey, D. V. and Paar, C.: Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms, Proc. of Crypto'98, LNCS 1462, Springer-Verlag, pp.472-485 (1998).

[3] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Eurocrypt'97*, LNCS, Springer-Verlag, Berlin, 1997.

[4] V. Shoup. A Proposal for an ISO Standard for Public Key Encryption (v.2.0). ISO/IEC JTC1/SC27, N2563, http://shoup.net/papers/, 2001 Sep.