International Journal of Cryptology Research 1 (1): 21-32 (2009)

Java Implementation for Identity-Based Identification

[1]Syh-Yuan Tan, [1]Swee-Huay Heng, [1]Bok-Min Goi, [1]Ji-Jian Chin and [2]SangJae Moon
[1]Multimedia University
[2]Kyungpook University
Email: syhyuan@yahoo.com

ABSTRACT

There are a lot of papers on cryptography implementation but mostly on encryption and signature schemes. In this paper, we provide the discussion on the implementation of five selected Identity-Based Identification (IBI) schemes, namely, the Schnorr IBI, the Bellare-Namprempre-Neven IBI, the Okamoto IBI, the Cramer-Shoup IBI, and the Guillou-Quisquater IBI. C programming language is a preferable platform in cryptography implementation due to its low running time, however, C is platform dependent and changes are needed each time the code is experimented on different operating systems. Thus, we opt for Java in implementing these selected schemes. Though the run time in Java is slightly higher (caused by Java Virtual Machine) than C, it is more convenient and flexible for us to try the same piece of code in different operating systems in future.

INTRODUCTION

An identification scheme assures one party (through acquisition of corroborative evidence) of both the identity of a second party involved, and that the second party was active at the time the evidence was created or acquired. In other words, identification protocol is an interactive process that enables a prover with a private key to identify himself to a verifier with the corresponding public key. Common applications of identification are Identity Card, ATM Card, Credit Card, E-voting, E-purse, etc. Meanwhile, Identity-based (ID-based) cryptography is a concept formalized by Shamir in 1984 [Sh85] where the public key is replaced by the user's public identity string, ID (name, email address, phone number etc.). The advantage of ID-based cryptography is no keys or certificates storage and data managing and searching operation are therefore eliminated. But the disadvantage is key escrow problem since it needs a trusted third party called the private key generator (PKG) to generate user's private key. The compromise of PKG's master-key is more disastrous than the compromise of the traditional Certification Authority's signing key. However, this key escrow feature is useful in some closed group operations.

In this paper, we provide the discussion on the implementation of five selected Identity-Based Identification (IBI) schemes, namely, the Schnorr IBI, the Bellare-Namprempre-Neven IBI, the Okamoto IBI, the Cramer-

Shoup IBI, and the Guillou-Quisquater IBI. These schemes are implemented using Java Cryptography Architecture (JCA), the Java security framework of Java Security API in Java Standard Edition 6.0 (J2SE6). JCA was designed under the following principles: implementation independence, implementation interoperability and algorithm extensibility. The implementation independence principle is significant in this implementation job as most of the needed algorithms were implemented inside provider. This means we need not code ourselves the key pair generators, hash (Message Digest), multiplicative inverse, etc. Until J2SE 6, JCA have not provided key generator algorithm for Elliptic Curve Cryptography (ECC) but Java does support the library of other providers for ECC or users can explicitly define their own favored parameters.

The simulations of all the probabilistic polynomial time algorithms of the chosen IBI schemes are run on a single computer with the following specifications: Intel Pentium M 1.6Ghz, 512RAM, Windows XP Service Pack 2. These simulations can sufficiently represent the real-world applications as all algorithms in the chosen IBI schemes are taken into consideration. The measured running time of the algorithms is the computation time of the schemes, which is easier to be controlled as compared to communication time. Communication time is not taken into consideration here as it is hardware and environment dependent. The run time for the single basic mathematical operations will also be shown besides the total execution time of each algorithm of the schemes. The result shown in the paper is the average of 1000 executions with random 512-bit input and timing is recorded in nanoseconds using Java provided nanoTime() function.

We organize the rest of the paper as follows. In Section 2, we give the definition of IBI and schemes chosen. In Section 3, we show the implementation result and followed by discussion in Section 4. Finally, we conclude in Section 5.

IDENTITY-BASED IDENTIFICATION

Definition

IBI was first rigorously defined and formalized in year 2004 independently by Kurosawa and Heng [KH04] and Bellare et. al. [BNN04]. It consist of three probabilistic polynomial time (PPT) algorithms, which are *setup algorithm* (S), *extract algorithm* (E), and *Identification Protocol* (P):

- Setup. S takes a security parameter 1^k and generates system parameters *params* and the *master-key*. The system parameters will be publicly known while *master-key* will be kept by PKG only.
- Extract. E takes as input the *params*, *master-key* and *ID*. It returns the *user private key*.
- Identification Protocol. P takes as input *params, ID, user private key* while V takes as input *params* and *ID*. After 3 move protocols which are *commit*, *challenge* and *response* between P and V, V makes the decision either accepts or reject the user.

Notion of Security

Three types of attacks are considered on the honest, private key equipped prover, [BNN04, KH04]:

- Passive attack – the adversary can eavesdrop and he is in possession of transcripts of conversations between the provers and the verifiers.
- Active attack – the adversary first plays the role of a cheating verifier, interacting with the provers several times before the impersonation.
- Concurrent attacks – the adversary first plays the role of a cheating verifier, interacting with the provers several times concurrently before the impersonation.

Selected IBI Schemes

i. *Schnorr IBI* [Heng04]

- Setup. On input 1^k, generate two large primes p and q of size k such that $q \mid (p-1)$, and an element $g \in Z_p$ of order q, where g is a generator of G, a subgroup of Z_{p^*} of order q. Choose a cryptography hash function $H : \{0,1\}^* \times G \times G \to Z_q$. Choose $s \in Z_q$ randomly and compute $v \equiv g^{-s} \bmod p$. The system parameters *params* = (p, q, g, v, H) and the *master-key* is s which is known to the PKG only.
- Extract. Given a public identity *ID*, choose $t \in Z_q$ randomly and compute $X \equiv g^t \bmod p, \quad \alpha \equiv H(ID, X, v), \quad Y \equiv t + s\alpha \bmod q.$ The user private key is (α, Y).
- Identification Protocol (secure in passive attack).

1. P first computes $X \equiv g^Y v^\alpha \bmod p$. P next chooses $r \in Z_q$ randomly and computes $x \equiv g^r \bmod p$. P finally sends (X, x) to V.
2. V chooses $c \in Z_q$ randomly and sends c to P.
3. P computes $y \equiv r + cY \bmod q$ and sends y to V.
4. V accepts if and only if $g^y \equiv x\left(X / v^\alpha\right)^c \bmod p$, where $\alpha = H(ID, X, v)$.

- Identification Protocol (secure in active and concurrent attack). The following protocol is executed $k = \log_2 q$ times in sequence:
  1. P first computes $X \equiv g^Y v^\alpha \bmod p$. P next chooses $r \in Z_q$ randomly and computes $x \equiv g^r \bmod p$. P finally sends (X, x) to V.
  2. V chooses $c \in \{0,1\}$ randomly and sends c to P.
  3. P computes $y \equiv r + cY \bmod q$ and sends y to V.
  4. V accepts if and only if $g^y \equiv x\left(X / v^\alpha\right)^c \bmod p$, where $\alpha = H(ID, X, v)$.

ii. *Okamoto IBI* [BNN04]

- Setup. On input 1^k, generate two large primes p and q of size k such that $q \mid (p-1)$, and an element $g \in Z_p$ of order q, where g is a generator of G, a subgroup of Z_p^* of order q. Choose a cryptography hash function $H : \{0,1\}^* \times G \times G \to Z_q$. Choose $x_1 x_2 \in Z_q$ randomly and compute $X \equiv g_1^{-x} g_2^{-x} \bmod p$. The system parameters *params* = (p, q, g_1, g_2, X, H) and the *master-key* is x_1, x_2 which is known to the PKG only.
- Extract. Given a public identity *ID*, choose $r_1, r_2 \in Z_q$ randomly and compute $R \equiv g_1^{r_1} g_2^{r_2} \bmod p$, $s_1 \equiv -r_1 - H(R \| I) \cdot x_1 \bmod q$, $s_2 \equiv -r_2 - H(R \| I) \cdot x_2 \bmod q$. The user private key is (R, s_1, s_2).
- Identification Protocol.
  1. P first computes $S \equiv g_1^{-s_1} g_2^{-s_2} \bmod p$. P next chooses $y_1, y_2 \in Z_q$ randomly and computes $Y \equiv g_1^{-y_1} g_2^{-y_2} \bmod p$. P finally sends (R, S, Y) to V.

2. V chooses $c \in Z_q$ randomly and sends c to P.
3. P computes $z_1 \equiv y_1 + cs_1 \bmod q$, $z_2 \equiv y_2 + cs_2 \bmod q$ and sends z_1, z_2 to V.
4. V accepts if and only if $Y \equiv g_1^{z_1} g_2^{z_2} S^c \bmod p$ and $R \equiv SX^{H(R\|ID)} \bmod p$.

iii. *Bellare-Namprempre-Neven IBI* [BNN04]

- Setup. On input 1^k, generate two large primes p and q of size k such that $q \mid (p-1)$, and an element $g \in Z_p$ of order q, where g is a generator of G, a subgroup of Z_p^* of order q. Choose a cryptography hash function $H: \{0,1\}^* \times G \times G \to Z_q$. Choose $x \in Z_q$ randomly and compute $X \equiv g^x \bmod p$. The system parameters $params = (p, q, g, X, H)$ and the *master-key* is x which is known to the PKG only.
- Extract. Given a public identity *ID*, choose $r \in Z_q$ randomly and compute $R \equiv g^y \bmod p, s \equiv r + H(R\|ID)\, x \bmod q$. The user private key is (R, s).
- Identification Protocol.
  1. P first computes $S \equiv g^s \bmod p$. P next chooses $y \in Z_q$ randomly and computes $Y \equiv g^y \bmod p$. P finally sends (R, S, Y) to V.
  2. V chooses $c \in Z_q$ randomly and sends c to P.
  3. P computes $z \equiv y + cs \bmod q$ and sends y to V.
  4. V accepts if and only if $g^z \equiv YS^c \bmod p$ and $S \equiv RX^{H(R\|ID)} \bmod p$.

iv. *Cramer-Shoup IBI* [Heng04]

- Setup. Two random k'-bit primes p and q are chosen, where $p = 2p' + 1$ and $q = 2q' + 1$, with both p' and q' prime. Let $N = pq$. Choose $h, x \in QR_N$ and $(k+1)$-bit prime b' randomly. Choose a collision-resistant hash function H. The system parameters $params = (N, h, x, b', H)$ and the *master-key* is (p, q).

- Extract. Given a public identity $ID \in \{0,1\}^*$, a random $(k+1)$ - bit prime $b \neq b'$ is chosen, and a random $y' \in QR_N$ is chosen. The equation $y^b \equiv xh^{H(x')} \bmod N \bmod N$ is solved for y, where x' satisfies the equation $\left(y'\right)^{b'} \equiv x'h^{H(ID)} \bmod N$. The user private key is $\left(b, y, y'\right)$.
- Identification Protocol (secure in passive attack).
  1. P chooses $s \in Z_q$ randomly and computes $t \equiv s^b \bmod N$. P sends $\left(b, y', t\right)$ to V.
  2. V chooses $c \in \{0,1,...,b-1\}$ randomly and sends c to P.
  3. P computes $z \equiv sy^c \bmod N$ and sends z to V.
  4. V accepts if and only if $z^b \equiv t\left(xh^{H(x')}\right)^c \bmod N$, where $x' \equiv \left(y'\right)^{b'} h^{-H(ID)} \left(\bmod N\right)$.
- Identification Protocol (secure in active and concurrent attack). The following protocol is executed for $k = \log p$ times:
  1. P chooses $s \in Z_q$ randomly and computes $t \equiv s^b \bmod N$. P sends $\left(b, y', t\right)$ to V.
  2. V chooses $c \in \{0,1\}$ randomly and sends c to P.
  3. P computes $z \equiv sy^c \bmod N$ and sends z to V.
  4. V accepts if and only if $z^b \equiv t\left(xh^{H(x')}\right)^c \bmod N$, where $x' \equiv \left(y'\right)^{b'} h^{-H(ID)} \left(\bmod N\right)$.

v. *Guillou-Quisquater IBI* [Heng04]

- Setup. Run RSA $\left(1^k\right)$ to obtain (N, a, b) where b is a large prime. Choose a cryptographic hash function $H : \{0,1\}^* \to Z_N{}^*$. The system parameters $params = (N, b, H)$ and the master-key is a which is know to the PKG only.
- Extract. Given a public identity *ID*, compute the user private key $d \equiv H\left(ID\right)^a \bmod N$.
- Identification Protocol.

1. P chooses $r \in Z^*_N$ randomly and sends $x \equiv r^b \bmod N$ to V.
2. V chooses $c \in \{0,1,...,b-1\}$ randomly and sends c to P.
3. P computes $y \equiv rd^C \bmod N$ and sends y to V.
4. V accepts if and only if that $y^b \equiv xH(ID)^C \bmod N$.

IMPLEMENTATION RESULTS

The following depicts the computational complexity of each scheme:

TABLE 1: Complexity comparison

	Schnorr [Heng04]	BNN [BNN04]	Okamoto [BNN04]	CS [Heng04]	GQ [Heng04]
Setup Cost	$2T_e + T_m$	T_e	$2T_e + T_m$	$T_m + T_a + 2T_{QR}$	$T_m + T_a + T_i$
Extract Cost	$6T_e + 4T_m + T_h + T_a$	$T_e + T_m + T_h + T_a$	$2T_m + 2T_h + 2T_a$	$4T_e + 2T_m + 2T_h + 2T_i$	$T_e + T_h$
Identification Cost	$6T_e + 4T_m + T_h + T_i + T_a$	$5T_e + 3T_m + T_h + T_a$	$8T_e + 7T_m + T_h + 2T_a$	$8T_e + 3T_m + 2T_h + T_i$	$4T_e + 2T_m + T_h$

T_e = exponentiation time T_h = hash time T_m = multiplication time

T_m = addition time T_i = multiplicative inverse time T_{QR} = quadratic residue time

TABLE 2: Operation Timing

Operations	Syntax	512-bit (ns)	1024-bit (ns)
Exponentiation*	modPow (BigInteger pow, BigInteger mod)	8,155,056	15,942,848
Multiplication	multiply (BigInteger val)	10,781	62,394
Multiplicative Inverse*	modInverse (BigInteger mod)	525,584	1,194,377
SHA-1	MessageDigest.getInstance (“SHA1”)	33,784	39,010
SHA-512	MessageDigest.getInstance (“SHA-512”)	58,730	883,922
Modular$	Mod (BigInteger mod)	26,132	54,956
Addition	Add (BigInteger val)	3,363	6,702
Quadratic Residue	jacobi(BigInteger val, BigInteger n)	6,053,811	6,280,951

*modulo is random 512-bit number $value size is the double of modulo bit-length

There are times where we wish to have all the operations finished executed before ending the final value with a single modular operation. However, this ideal case cannot be implemented because in JCA, syntax of exponentiation comes together with a modular operation as shown in Table 2. This cannot be avoided and thus affect the efficiency. The following are parameters used in each scheme:

TABLE 3 : Schemes Parameters

Scheme	Parameters
Schnorr, BNN, Okamoto	Setup: DSA KeyPairGenerator
	Hash: SHA-1
	ID: user@mmu.edu.my
CS	Setup: RSA KeyPairGenerator
	Hash: SHA-512
	Quadratic Residue: Jacobi symbol
	ID: user@mmu.edu.my
GQ	Setup: RSA KeypairGenerator
	Hash: SHA-512
	ID: user@mmu.edu.my

J2SE6 does not provide algorithm to find a quadratic residue modulo neither a prime nor a composite number. In order to find the quadratic residue modulo N (N is composite number) in CS scheme, we coded Jacobi symbol algorithm as in [Fis06]. For other details of parameters mentioned, kindly refer to [Java2]. We run the five schemes for 1000 times using the parameters above with a 512-bit key and 1024-bit key independently on Intel Pentium M 1.6 GHz, 512RAM, Windows XP Service Pack 2. We then average the running time.

TABLE 4*:* Timing for 512-bit

	Setup (ns)	Extract (ns)	Identification [P] (ns)	Identification [A&C] (ns)
Schnorr	4,891,679	1,272,181	12,826,772	1,692,239,880
BNN	4,556,400	1,290,435	9,112,274	9,112,274
Okamoto	7,399,199	2,367,228	15,156,250	15,156,250
CS	552,521,647	119,084,865	96,400,380	8,452,845,999
GQ	84,823,200	6,849,624	1,149,553	1,149,553

ns – nanosecond A&C – active and concurrent attack
P – passive attack

TABLE 5: Timing for 1024-bit

	Setup (ns)	Extract (ns)	Identification [P] (ns)	Identification [A&C] (ns)
Schnorr	16,059,077	4,232,809	43,624,176	5,351,031,031
BNN	16,563,530	4,443,884	33,784,060	33,784,060
Okamoto	25,222,381	8,359,174	52,977,404	52,977,404
CS	7,829,626,400	1,608,030,778	952,885,593	441,142,435,428
GQ	516,212,870	44,216,585	3,503,753	3,503,753

ns – nanosecond A&C – active and concurrent attack
P – passive attack

From the results, GQ is the most efficient scheme and CS is the slowest scheme. However, it is reasonable since CS scheme is proven secure in standard model [CS00] and thus its tradeoff is the performance.

DISCUSSIONS

Comparing the complexity and running time, we notice that even the complexity is about the same, timing result may vary a lot. The *Extract* and *Identification Protocol* of Schnorr scheme varies on only a single inverse operation but the timing has the difference of 11,554,591ns for 512-bit key. This is because inverse operation is costly since it uses extended Euclidean algorithm. However, the main concern is the bit-length of numbers instead of the amount of operations. For instance, the *Extract* in GQ scheme has only 2 operations but it takes more time than its *Identification Protocol*. The value α, $H(ID)$ and N in *Extract* has bit-length about 512 on average while value b in *Identification Protocol* is fixed to be 65537 with bit-length of 17 only. Consequently, bit-length of exponent value c in *Identification Protocol* would be much smaller (compare to α) and thus the execution time of modular operation (with respect to N) is reduced.

In Schnorr scheme, since the size of q is set to 160-bit despite of the size of p, hash function SHA-1 is used in this scheme as its output is always 160-bit. In real world application, SHA-512 is recommended (SHA-1 is theoretically broken [CR06]) but 512-bit hash value needs to go through a modular operation for q is only 160-bit. For the scheme to be secure against active and concurrent attack, the *Identification Protocol* is run for $k = \log_2 q$ times sequentially and $c \in \{0,1\}$ [Heng04]. In BNN scheme, the hash function issue is the same as in Schnorr scheme. Since it is proven to be secure against active and concurrent attacks as in scheme secured against passive attack [BNN04], no change is needed on *Identification Protocol*. Similar with BNN scheme, no change is needed on *Identification Protocol* for Okamoto scheme.

For CS scheme, as suggested in [CS00], optimization can be done by letting $x \equiv h^a \bmod N$ for a random number $\alpha \bmod p'q'$, where α is stored in the secret key. Thus $y^b \equiv xh^{H(x')} \bmod N$ will become $y^b \equiv h^{a+H(x')} \bmod N$. Let d be the inverse of $b \bmod p'q'$, then $y \equiv h^{da+dH(x')} \bmod N$. This saves time by replacing $T_m + T_e$ with $T_m + T_a$ in *Extract* and replacing T_m with T_m in *Identification Protocol*. For the scheme to secure against active and concurrent attack, the identification protocol is run for k times in sequence where $k = \log_2 p$ or equivalently $k = \log_2 q$ and $c \in \{0,1\}$ [Heng04]. Similar with BNN and Okamoto schemes, GQ scheme is proved to be secure against active and concurrent attacks as in the scheme secure against passive attack [Heng04] and thus no change is needed on its *Identification Protocol*.

CONCLUSION

We had provided a brief overview of IBI and obtain the running time for Schnorr, Bellare-Namprempre-Neven, Okamoto, Cramer-Shoup and Guillou-Quisquater IBI. We had also discussed on their implementation and performance.

ACKNOWLEDGEMENT

This research was supported by the Malaysia e-Science Fund (01-02-01-SF0032) and the Ministry of Knowledge Economy (MKE) of Korea, under the ITRC support program supervised by the IITA (IITA-2008-C1090-0801-0016).

REFERENCES

[BNN04] M. Bellare, C. Namprempre, G Neven. 2004. Security Proofs for Identity-Based Identification and Signature Schemes. *Advances in Cryptology – EUROCRYPT 2004*, LNCS 3027.

[CR06] Christophe De Cannière, Christian Rechberger. 2006. Finding SHA-1 Characteristics: General Results and Applications. *ASIACRYPT 2006*, LNCS 4284, 1-20.

[CS00] R. Cramer, V. Shoup. 2000. Signature Schemes Based on the Strong RSA Assumption. *IBM Research Report RZ 3038*.

[Ed04] Kossi D. Edoh. 2004. Elliptic Curve Cryptography: Java Implementation. *InfoSecCD Conference '04*

[Fis06] M. J. Fischer. 2006. Cryptography and Computer Security: *Lecture Notes 13*, Yale University: Department of Computer Science, http://zoo.cs.yale.edu/classes/cs467/2006f/attach/ln13.html.

[GQ88] L. Guilloi and J. J. Quaisquater. 1988. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. *Advances in Cryptology – CRYPTO '88*, LNCS Vol. 403.

[Heng04] Heng Swee Huay. 2004. Design and Analysis of Some Cryptographic Primitives. PhD Thesis, *Tokyo Institute of Technology*.

[Hol06] David Holme. 2006. Inside the Hotspot VM: Clocks, Timer and Scheduling Event - Part 1 – Windows. http://blogs.sun.com/dholmes/entry/ inside_the_hotspot_vm_clocks.

[JS01] P.K. Janbandhu, M.Y. Siyal. 2001. Novel biometric digital signatures for Internet-based applications. *Information Management & Computer Security*, 9:5, 205-212.

[Java1] Java™ Cryptography Architecture (JCA) Reference Guide for Java™ Platform Standard Edition 6. http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[Java2] Java™ Cryptography Architecture Standard Algorithm Name Documentation for Java™ Platform Standard Edition 6. http://java.sun.com/javase/6/docs/technotes/guides/security/StandardNames.html.

[KH04] Kaoru Kurosawa, Swee-Huay Heng. 2004. From Digital Signature to ID-based Identification/Signature. *Public Key Cryptography – PKC'04*, LNCS 2947, 248-261.

[KH05] Kaoru Kurosawa, Swee-Huay Heng. 2005. Identity-Based Identification Without Random Oracles. *ICCSA 2005, LNCS 3481*, 603-613.

[Mao03] Wenbo Mao. 2003. Modern Cryptography: Theory and Practice. *Prentice Hall Professional Technical Reference*, 186-192.

[MOV97] A. J. Menezes, P.C van Oorschot and S. A. Vanstone. 1997. *Handbook of Applied Cryptography*. CRC Press.

[Nig06] Jeremy S. Nightingale. 2006. Comparative Analysis of Java Cryptographic Libraries for Public Key Cryptography. George Mason University: Department of Electrical and Computer Engineering, http://ece.gmu.edu/courses/ECE746/project/reports_2006/JAVA_MULTIPRECISION_report.pdf

[Oka92] T. Okamoto. 1992. Provably secure and practical identification schemes and corresponding signature schemes. *Advances in Cryptology – CRYPTO '92*. LNCS Vol. 740.

[Sc91] C. Schnorr. 1991. Efficient signature generation by smart cards. *Journal of Cryptology*, vol. 4, pp. 161-174.

[SCA06] Michael Scott, Neil Costigan, Wesam Abdulwahab. 2006. Implementing Cryptographic Pairings on Smartcards. *CHES '06*, LNCS 4249, 134-147.

[Sh85] Adi Shamir. 1985. Identity-based Cryptosystem and Signature Schemes. *Advances in Cryptology – Crypto '84*, LNCS vol. 196, 47-53.