LETTER Special Section on Information and Communication System Security

Efficient Flexible Batch Signing Techniques for Imbalanced Communication Applications*

Taek-Young YOUN[†a)], **Young-Ho PARK**[††b)], **Taekyoung KWON**[†††c)], **Soonhak KWON**[††††d)], *Nonmembers*, *and* **Jongin LIM**[†††††e)], *Member*

SUMMARY Previously proposed batch signature schemes do not allow a signer to generate a signature immediately for sequentially asked signing queries. In this letter, we propose flexible batch signatures which do not need any waiting period and have very light computational overhead. Therefore our schemes are well suited for low power devices.

key words: *signature, batch signing, imbalanced communication*

1. Introduction

Batch signature is a cryptographic tool for signing multiple messages at a time with a cost of single signing. Until now three batch signing techniques have been proposed in [1], [6] and [2]. Among them, two schemes, in [1] and [6], are generic techniques whose basic idea is to compress given messages using cryptographic hash function and to sign it using well-known signature schemes such as DSA [5] and Schnorr's signature [7]. A difference between two techniques is the way to compress messages. One method concatenates hash of given messages and hashing it once more. Another method constructs a hash tree for given messages based on the Merkle tree. A verifier computes the compressed message when he verifies a batch signature. Differ from the generic constructions, in [2], a specific batch signature is proposed based on RSA, which computes a signature for many messages and separates the signature into distinct signatures for each message. In this case, a signer uses several public keys, and the number of public keys corresponds to the size of batch.

Above-mentioned techniques do not allow a signer to generate a batch signature for sequentially asked signing queries. To fully enjoy the merit of batch signature, a requester should wait until sufficient requests are accumulated, but it makes the communication inefficient. We define a batch signature as *flexible* if a signer can generate a batch signature for sequential requests without a waiting period. The existing batch signing techniques are not flexible, and so they can not fully enjoy the merit of batch signing for sequentially given requests. In [4], a fast DLP-based signing strategy is proposed which generate many signature efficiently by computing many exponentiations simultaneously. Though the technique can generate several signatures simultaneously, it is not a batch signature but a batch exponentiation technique which efficiently computes multiple exponentiations by computing them simultaneously.

In this letter, we propose a flexible batch signature which requires one exponentiation for six sequential signing requests, and one multiplication for each request. Since our signature is flexible, on average, a signer can respond to sequentially asked requests within 41 multiplications. Moreover, our scheme use a fixed base, and so we can respond to a signing query within 23 multiplications by adopting the comb-method [3]. We also propose a variant of our scheme using well-known Merkle tree. Though the cost of verification increases, our batch signatures can be useful for an imbalanced communication since they are flexible. One example is the applications for low power devices such as mobile phone and PDA which have limited electric power. In this case, it is important to reduce the use of electric power. If a low power device generates a signature using flexible batch signature to prove its identity, they can reduce the amount of power used for generating an authentication data. Another example is the server application where the server should generate huge amount of signatures for its clients. If the server uses a flexible batch signature, it can respond to the requests asked by its clients more promptly.

2. New Batch Signatures

We provide a flexible batch signature scheme and a variant of it. Recall that, by a flexible batch signature, we mean that it provides batch signing for n sequential requests as n-batch signature without a waiting period.

Manuscript received August 3, 2007.
Manuscript revised December 6, 2007.

[†]The author is with Graduate School of Information Management and Security, Korea University, Seoul, Korea.
[††]The author is with Information Security Systems, Sejong Cyber University, Seoul, Korea.
[†††]The author is with School of Computer Engineering, Sejong University, Seoul, Korea.
[††††]The author is with the Dept. of Mathematics, Sungkyunkwan University, Suwon, Korea.
[†††††]The author is with Graduate School of Information Management and Security, Korea University, Seoul, Korea, Corresponding author.

*This work was supported by grant No.R01-2005-000-11261-0 from Korea Science and Engineering Foundation in Ministry of Science & Technology.

a) E-mail: taekyoung@cist.korea.ac.kr
b) E-mail: youngho@cybersejong.ac.kr
c) E-mail: tkwon@sejong.ac.kr
d) E-mail: shkwon@skku.edu
e) E-mail: jilim@korea.ac.kr

DOI: 10.1093/ietisy/e91–d.5.1481

2.1 Flexible Batch Signature 1 (FBS1)

We propose a signature which provides flexible batch signing for multiple messages. We describe the signature for the case where two messages are sequentially requested, i.e., we describe a 2-batch signature.

Setup(k): Let p be a prime number such that $q_1|p-1$ and $q_2|p-1$ for two primes q_1 and q_2 where $|q_1| = |q_2| = k$ for some security parameter k. Let g_1 and g_2 be two generators of order q_1 and q_2, respectively. Let $g = g_1^{\kappa_1} g_2^{\kappa_2} \pmod p$, where $\kappa_1 = q_2^{-1} \pmod{q_1}$ and $\kappa_2 = q_1^{-1} \pmod{q_2}$. Note that, by definition, $g_1^{q_1} = 1 \pmod p$ and $g_2^{q_2} = 1 \pmod p$. Then, $g_1 = g^{q_2} \pmod p$, since the following relation holds: $g^{q_2} = (g_1^{\kappa_1} g_2^{\kappa_2})^{q_2} = g_1^{q_2\kappa_1} g_2^{q_2\kappa_2} = g_1^{q_2\kappa_1} = g_1 \pmod p$. Similarly, $g_2 = g^{q_1} \pmod p$. Let $h_1 : \{0,1\}^* \to \mathbb{Z}_{q_1}$ and $h_2 : \{0,1\}^* \to \mathbb{Z}_{q_2}$ be two cryptographic hash functions that map an arbitrary string to an element of $\mathbb{Z}_{q_1}$ and $\mathbb{Z}_{q_2}$, respectively. For given k, $params = \{p, q_1, q_2, g, h_1, h_2\}$ is the system parameter.

KeyGen(params): An user chooses two secret values $x_1 \in \mathbb{Z}_{q_1}$ and $x_2 \in \mathbb{Z}_{q_2}$, and computes $y_1 = g^{q_2 x_1} \pmod p$ and $y_2 = g^{q_1 x_2} \pmod p$. Then the public key for the user is computed as $y = y_1^{\kappa_1} y_2^{\kappa_2}$, and the corresponding secret key is $sk = \{x_1, x_2\}$. Note that, $y_1 = g_1^{x_1} \pmod p$, and so the following relation holds: $y_1^{q_1} = g_1^{x_1 q_1} = (g_1^{q_1})^{x_1} = 1 \pmod p$. Similarly, $y_2^{q_2} = 1 \pmod p$. So, $y^{q_2} = y_1 \pmod p$ and $y^{q_1} = y_2 \pmod p$.

BatchSign(params, sk, m_1, m_2): Note that two messages are not required to be requested simultaneously. When two messages, m_1 and m_2, are given to sign, a signer chooses a random $r \in \{0,1\}^k$, and computes $\beta = g^r \pmod p$. He computes $\alpha_1 = x_1 h_1(m_1||\beta) + r \pmod{q_1}$ and $\alpha_2 = x_2 h_2(m_2||\beta) + r \pmod{q_2}$. Then, $\sigma_1 = (\alpha_1, \beta)$ and $\sigma_2 = (\alpha_2, \beta)$ are signatures for m_1 and m_2, respectively.

Verify(params, pk, σ_i, m_i): When a signature $\sigma_i = (\alpha_i, \beta)$ for a message m_i is given to verify, a verifier checks whether the following equality holds: $(y^{h_i(m_i||\beta)} \beta / g^{\alpha_i})^{q_{3-i}} \stackrel{?}{=} 1 \pmod p$. The same condition can be verified using the following equality: $(g^{q_{3-i}})^{\alpha_i} \stackrel{?}{=} (y^{h_i(m_i||\beta)}\beta)^{q_{3-i}} \pmod p$. If the condition holds, the verifier accepts σ_i as a valid signature. We can check the correctness of the verification equation as following: $(y^{h_i(m_i||\beta)}\beta)^{q_{3-i}} = y_i^{h_i(m_i||\beta)} (g^r)^{q_{3-i}} = (g_i^{x_i})^{h_i(m_i||\beta)} g_i^r = g_i^{x_i h_i(m_i||\beta)+r} = (g^{q_{3-i}})^{\alpha_i} \pmod p$.

FBS1 supports up to 6-batch signing in a natural way. For k_1-bit prime modulus, we can choose at most $\lfloor k_1/k_2 \rfloor$ generators that have distinct k_2-bit prime order. Especially, for 1024-bit prime, we can choose at most six ($= \lfloor 1024/160 \rfloor$) generators of 160-bit prime order. System parameter for 6-batch signature is then $params = \{p, Q, g, H\}$ where p is a 1024-bit prime modulus, q_i is a 160-bit prime such that $q_i|p-1$, g_i is a generator of order q_i, $h_i : \{0,1\} \leftarrow \mathbb{Z}_{q_i}$ is a cryptographic hash function, $Q = \{q_1, \ldots, q_6\}$, $g = \prod_{i=1}^6 g_i^{\kappa_i} \pmod p$ for $\kappa_i = ((\prod_{j=1}^6 q_j)/q_i)^{-1} \pmod{q_i}$ and $H = \{h_1, \ldots, h_6\}$. Existence of such parameters will be shown in Sect. 4.1. For 6-batch signature, a signer chooses six secrets $x_i \in \mathbb{Z}_{q_i}$ for $i \in \{1, \ldots, 6\}$, and computes $y_i = (g^{(\prod_{j=1}^6 q_j)/q_i})^{x_i}$ and the public $y = \prod_{i=1}^6 y_i^{\kappa_i}$. The corresponding secret key is $\{x_1, \ldots, x_6\}$. For given six messages, the signer computes $\alpha_i = x_i h_i(m_i||\beta) + r \pmod{q_i}$ and $\beta = g^r \pmod p$ for $i \in \{1, \ldots, 6\}$. For given $\sigma_i = (\alpha_i, \beta)$, a verifier checks the following: $(y^{h_i(m_i||\beta)}\beta / g^{\alpha_i})^{(\prod_{j=1}^6 q_j)/q_i} \stackrel{?}{=} 1 \pmod p$.

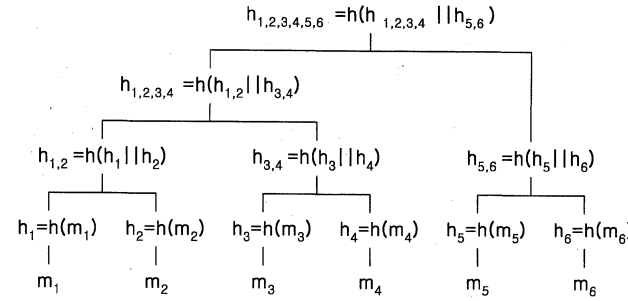

Fig. 1 Construction of hash tree for six messages.

2.2 Flexible Batch Signature 2 (FBS2)

We propose a variant of FBS1 using Merkle's hash-tree technique. We define two notions related to hash-tree. Let *root* be a value assigned to the highest parent node for a hash-tree, and *covering set* for a message be a set of values being used to generate the root with the given message. In Fig. 1, $h_{1,2,3,4,5,6}$ is the root, and $CS = \{h_{1,2}, h_4, h_{5,6}\}$ is the covering set for m_3. From m_3 and CS, the root can be computed as following: $h_{1,2,3,4,5,6} = h(h(h_{1,2}, h(h(m_3), h_4)), h_{5,6})$.

Setup(k) and *KeyGen(params):* same to FBS1

BatchSign(params, sk, M_1, M_2): We assume that each set of messages, M_1 and M_2, are requested simultaneously. Let $M_1 = \{m_{1,i_1} | i_1 \in [1, b_1]\}$ and $M_2 = \{m_{2,i_2} | i_2 \in [1, b_2]\}$ where b_1 and b_2 are the number of messages in M_1 and M_2, respectively. When M_1 and M_2 are given to sign, a signer constructs two hash-tree for each message set as shown in Fig. 1. Let m_1 and m_2 be the root of hash trees for M_1 and M_2, respectively. The signer chooses a random $r \in \{0,1\}^k$, and computes $\beta = g^r \pmod p$. He computes $\alpha_1 = x_1 h_1(m_1||\beta) + r \pmod{q_1}$ and $\alpha_2 = x_2 h_2(m_2||\beta) + r \pmod{q_2}$. The signature for a message $m_{i,j}$ is then $\sigma_{i,j} = (\alpha_i, \beta, CS_{i,j})$ with $i \in \{1,2\}$ and $j \in [1, b_i]$, where $CS_{i,j}$ is the covering set for $m_{i,j}$.

Verify(params, pk, $\sigma_{i,j}$, $m_{i,j}$): When a signature $\sigma_{i,j}$ is given, a verifier recovers m_i from $m_{i,j}$ and $CS_{i,j}$, and he checks the following equality: $(y^{h_i(m_i||\beta)}\beta / g^{\alpha_i})^{q_{3-i}} \stackrel{?}{=} 1 \pmod p$. If the condition holds, he accepts $\sigma_{i,j}$ as a valid signature.

3. Analysis

Let us denote the Schnorr's signature, simple batch signing technique [1], tree-based batch signing technique [6] and the batch exponentiation method [4] as SCS, SBS, TBS and BEXP, respectively.

3.1 Security

One of the main difference between our constructions and the previous schemes is the use of same random value for different signatures in our case. For conventional signatures, such as SCS and DSA, the use of the same random r makes an adversary can compute the secret key. Though our schemes use the same random for different messages in a same batch, a similar attack as in SCS cannot be applied. We explain it for the case of 6-batch signing since it reveals much of the information to an adversary compared with the case of batch signature with fewer requests. When 6-batch signing is executed, an adversary can obtain the following values:

$$x_1h_1(m_1||\beta)+r \pmod{q_1},\ x_2h_2(m_2||\beta)+r \pmod{q_2},$$
$$x_3h_3(m_3||\beta)+r \pmod{q_3},\ x_4h_4(m_4||\beta)+r \pmod{q_4},$$
$$x_5h_5(m_5||\beta)+r \pmod{q_5},\ x_6h_6(m_6||\beta)+r \pmod{q_6},$$

where $\beta = g^r \pmod{p}$. In general, an adversary can recover the secret key by solving a system of equations generated by the same r.[†] For our signatures, an adversary also can recover secret keys when the above system of equations are solvable. However, we use different key for each signature and each equation is defined over different modulus, and so the given system of linear equations is hard to solve.

The other difference between our constructions and the previous schemes is the number of secret keys for a public key y. In general, a public key is assigned to a secret key, however, in our scheme, single public key is published for several secret keys. Though single value y is published as a public key, we can use different public key for each secret key. Each public key can be uniquely derived from y. In our schemes, the virtual public key for x_i is $y_i = (g^{(\prod_{j=1}^{6} q_j)/q_i})^{x_i}$. Note that, $y = \prod_{i=1}^{6} y_i^{\kappa_i}$. Then we can easily derive y_i from y as following: $y^{\gamma_i} = \prod_{l=1}^{6} y_l^{\kappa_l\gamma_i} = y_i^{\kappa_i\gamma_i} = y_i$, where $\gamma_i = (\prod_{j=1}^{6} q_j)/q_i$. Note that, $y_l^{\kappa_l\gamma_i} = 1 \pmod{p}$ for $l \neq i$, since $q_l|\gamma_i$ and $y_l^{q_l} = 1 \pmod{p}$. Each public key can be uniquely reconstructed from y by using the above equation. Hence, the use of single public key for different secret keys is not a problem in our schemes.

3.2 Efficiency

In Table 1, we compare two proposed signatures with SCS and existing DLP-based batch signatures, SBS and TBS. We also compare our schemes with the signing technique based on BEXP, since its purpose is similar to a batch signature. From now, we call it as BEXPS. We compare their efficiency in terms of the number of modular exponentiation. We ignore the cost of hash evaluation, addition and multiplication, since their computational costs are negligible compared to that of exponentiation. Let M be the cost of multiplication, E be the cost of exponentiation (with 160-bit exponent), n be the number of signing requests and n_b be the number of batchs. Let n_k be the constant which measure the efficiency of BEXP. Note that, the cost of exponentiation with $160t$-bit exponent is tE. We can choose m at most 6. In Table 1, the character "–" means that the corresponding scheme is not a batch signature, and so to discuss its flexibility is meaningless.

Table 1 Efficiency comparison of the efficiency and property.

	Sign	Verification	Flexibility
SCS	nE	2E	–
SBS/TBS	n_bE	2E	X
BEXPS	n/n_kE	2E	–
FBS1	n/mE	(1+m)E	O
FBS2	n_b/mE	(1+m)E	O

Table 2 Efficiency of batch exponentiation (See Table 2 in [4] for detailed explanation).

Size of Batch Exp	Sign		Verification	n_k
	Cost	Storage	Cost	
2	141M	316B	2E	1.70
3	103M	652B	2E	2.33
4	85M	1324B	2E	2.84
12	58M	3844B	2E	4.15
36	49M	11404B	2E	4.90
108	46M	34084B	2E	5.22

For $m = 6$, FBS1 and FBS2 reduce the average signing cost by 83% compared with SCS and TBS, respectively. For an exponentiation with 160-bit exponent, 240 multiplications are required. Hence, on average, a signer can respond to a signing request within $41 = (240 + 6)/6$ multiplications, since one exponentiation and six multiplications are required for six signature requests.

The efficiency of BEXP depends on n_k. As seen in Table 2, average signing cost can be reduced to 46M if 108 exponentiations are simultaneously computed using 34,084B of storage. If we use the comb-method [3], FBS1 can generate a signature with 23M using 405B of storage, and so FBS1 is more efficient than the signature generation based on BEXP. Note that, BEXP can not adopt the comb-method.

4. Implementational Considerations

4.1 Parameter Selection

In this section, we provide an algorithm to generate a prime p such that $q_i|p-1$ for six distinct primes q_i. Let $\mathbb{P}_k$ be the set of primes in $[2^k, 2^{k+1}]$. $a \leftarrow b$ means that b is assigned to a. $a \leftarrow_R \mathbb{S}$ means that a is randomly chosen in a set $\mathbb{S}$.

[†]To resistance the attack, previous signatures, including SCS and DSA, use different random values for different signatures.

ALGORITHM 1
GENERATION OF PRIME p FOR 6-BATCH SIGNATURE

Step 1. for(i=1; i<7; i++) $q_i \leftarrow_R \mathbb{P}_{160}$
Step 2. $Q \leftarrow \prod_{i=1}^{6} q_i$
Step 2. $q \leftarrow_R \mathbb{Z}$ such that $|2qQ + 1| = 1024$
Step 3. $p \leftarrow 2q \prod_{i=1}^{6} q_i + 1$
Step 4. if(p is prime) returns $q_1, q_2, q_3, q_4, q_5, q_6$ and q
else go to Step 1

Other parameters can be derived from the choice of p. Though the given algorithm is specifically designed to generate 1024-bit prime, it is trivial to extend it to general cases. We generate the parameters using Windows XP, Pentium 4, Intel CPU 1.2 GHz, 1 GB RAM. The following is an example.

$q_1 = 2^{160}$+23B54F501274F91B591D19A27A19FA237E4F59AB
$q_2 = 2^{160}$+9A27E414F91BA19F5012754FD591A237F591C387
$q_3 = 2^{160}$+5012754F4F91B59127A19F501274F91B591A23B7
$q_4 = 2^{160}$+C23B54FD19A2F50127E4F5914F91B591A2377A57
$q_5 = 2^{160}$+4F5919A22741C23B54591A237EFDF91B7A19F643
$q_6 = 2^{160}$+41274F91B9A27A19F5A237E4F595911C23B551AF
q =19A27A19F2741FA5

4.2 Secure Storage

In our signatures, several private keys are used to support batch signing. So, more secure storage is required than other schemes. To cover the shortcoming, we securely store a secret seed s_{key} rather than store all secret keys, and derive six keys from the seed using a random number generating function. Then we can reduce the size of secure storage to store many private keys using k-bit secure storage to store the seed. For example, we compute $6k$ bit $K = x_1||x_2||x_3||x_4||x_5||x_6$ using the random generating function, and use x_1, x_2, x_3, x_4, x_5 and x_6 as six distinct secret keys.

We store and reuse a random r, and a private key can be revealed if the random is exposed to and adversary. Hence, r should be securely stored, and this may cause the implementation more complicate. To solve this problem, we derive $(6k + k')$-bit random string from the seed and use the least significant k'-bit of the random string to encrypt r. Note that, an adversary who obtains the ciphertext can not recover r, and so a long-term secret does not revealed.

5. Conclusion

In this letter, we proposed a flexible batch signature scheme and a variant of it, FBS1 and FBS2, respectively. Our schemes suit for imbalanced communication applications such as low power device applications and a server application where it responds to huge amount of signing requests asked by clients.

References

[1] W.C. Cheng, C.-F. Chou, and L. Golubchik, "Performance of batch-based digital signatures," Proc. 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS'02), pp.77–85, 2002.
[2] A. Fiat, "Batch RSA," Advances in Cryptology - Proc. Crypto'89, Lecture Notes in Computer Science, vol.435, pp.175–185, 1989.
[3] C. Lim and P. Lee, "More flexible exponentiation with precomputation," Advances in Cryptology - CRYPTO'94, Lecture Notes in Computer Science, vol.839, pp.239–252, Springer-Verlag, 1994.
[4] D. M'Raïhi and D. Naccache, "Batch exponentiation: A fast DLP-based signature generation strategy," Proc. 3rd ACM Conference on Computer and Communications Security (CCS'96), pp.58–61, 1996.
[5] NIST, "Digital signature standard (DSS)," FIPS Federal Register, vol.56, no.169, 1991.
[6] C.J. Pavlovski and C. Boyd, "Efficient batch signature generation using tree structures," Proc. International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99), pp.70–77, City University of Hong Kong Press, 1999.
[7] C.P. Schnorr, "Efficient identification and signatures for smart cards," Advances in Cryptology - Proc. EUROCRYPT'89, Lecture Notes in Computer Science, vol.435, pp.239–252, 1989.