Shortened Digital Signature, Signcryption and Compact and Unforgeable Key Agreement Schemes [*]

Yuliang Zheng

The Peninsula School of Computing and Information Technology
Monash University, McMahons Road, Frankston
Melbourne, VIC 3199, Australia
Email: `yuliang@pscit.monash.edu.au`
Phone: +61 3 9904 4196, Fax: +61 3 9904 4124

July 9, 1998

[*]A submission to *IEEE P1363a: Standard Specifications for Public-Key Cryptography: Additional Techniques*

Summary

This submission consists of three separate parts. Although these parts are technically related, each addresses a different issue in cryptography and can serve as an independent contribution to the standard. Schemes described in this submission can play a role complementary to those designed in a different approach. A common feature of these schemes is that they all attemp to minimize computational efforts and communication overhead involved in a cryptographic operation.

- In Part I, two slightly modified versions of the Digital Signature Standard (DSS) are presented. Both modified signature schemes are more efficient than DSS and admit provable security in the model of Pointcheval and Stern.
- This is followed by Part II where two signcryption schemes are described. A major advantage of signcryption schemes is that they fulfill both the functions of digital signature and public key encryption in a single step, and with a cost, both in terms of modular exponentiation and message overhead, significantly smaller than that required by "signature followed by encryption".
- Finally in Part III, a number of compact, non-repudiatable and unforgeable key agreement schemes/protocols are presented. All these protocols are based on the signcryption schemes.

The shortened signature, signcryption and key agreement schemes can all be extended to similar schemes on secure elliptic curves. This is outlined in Appendix A. Discussions on the computational efficiency of the schemes are dependent on a techique proposed by Shamir that allows fast computation of the product of multiple exponentials with the same modulo. This technique is briefly described in Appendix B.

The submission is based on the following publications whose on-line versions are available at `http://www.pscit.monash.edu.au/~yuliang/`

1. Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption). In *Advances in Cryptology - CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179, Berlin, New York, Tokyo, 1997. Springer-Verlag.
2. Y. Zheng. Signcryption and its applications in efficient public key solutions (invited talk). In *Information Security — Proceedings of 1997 Information Security Workshop (ISW'97)*, volume 1396 of *Lecture Notes in Computer Science*, pages 291–312, Berlin, New York, Tokyo, 1998. Springer-Verlag.
3. Y. Zheng and H. Imai. Compact and unforgeable session key establishment over an ATM network. In *Proceedings of IEEE INFOCOM'98*, pages 411–418, San Francisco, 1998. IEEE.
4. Y. Zheng and H. Imai. Efficient signcryption schemes on elliptic curves. In *Proceedings of the IFIP 14th International Information Security Conference (IFIP/SEC'98)*, Vienna and Budapest, August 1998.

Contents

List of Figures

List of Tables

Part I
Shortened Digital Signature Schemes

1 Description

ElGamal digital signature scheme [21] involves two parameters public to all users:

1. p: a large prime.
2. g: an integer in $[1, \ldots, p-1]$ with order $p-1$ modulo p.

User Alice's private key is an integer x_a chosen randomly from $[1, \ldots, p-1]$ with $x_a \not| (p-1)$ (i.e., x_a does not divide $p-1$), and her public key is $y_a = g^{x_a} \bmod p$.

Alice's signature on a message m is composed of two numbers r and s:

$$\begin{aligned} r &= g^x \bmod p \\ s &= (hash(m) - x_a \cdot r)/x \bmod (p-1) \end{aligned}$$

where $hash$ is a one-way hash function, and x is chosen independently at random from $[1, \ldots, p-1]$ with $x \not| (p-1)$ every time a message is to be signed by Alice. Given (m, r, s), one can verify whether (r, s) is Alice's signature on m by checking whether $g^{hash(m)} = y_a^r \cdot r^s \bmod p$ is satisfied.

Since its publication in 1985, ElGamal signature has received extensive scrutiny by the research community. In addition, it has been generalized and adapted to numerous different forms (see for instance [49, 10, 40, 42] and especially [27] where an exhaustive survey of some 13000 ElGamal based signatures has been carried out.) For most ElGamal based schemes, the size of the signature (r, s) on a message is $2|p|$, $|q| + |p|$ or $2|q|$, where p is a large prime and q is a prime factor of $p-1$. The size of an ElGamal based signature, however, can be reduced by using a modified "seventh generalization" method discussed in [27]. In particular, we can change the calculations of r and s as follows:

1. Calculation of r — Set $r = hash(k, m)$, where $k = g^x \bmod q$ (or $k = g^x \bmod (p-1)$ if the original r is calculated modulo $(p-1)$), x is a random number from $[1, \ldots, q-1]$ (or from $[1, \ldots, p-1]$ with $x \not| (p-1)$), and $hash$ is a one-way hash function such as Secure Hash Standard or SHS [41] and HAVAL [58].
2. Calculation of s — For an *efficient* ElGamal based signature scheme, the calculation of (the original) s from x_a, x, r and optionally, $hash(m)$ involves only simple arithmetic operations, including modular addition, subtraction, multiplication and division. Here we assume that x_a is the private key of Alice the message originator. Her matching public key is $y_a = g^{x_a} \bmod p$. We can modify the calculation of s in the following way:
   (a) If $hash(m)$ is involved in the original s, we replace $hash(m)$ with a number 1, but leave r intact. The other way may also be used, namely we change r to 1 and then replace $hash(m)$ with r.
   (b) If s takes the form of $s = (\cdots)/x$, then changing it to $s = x/(\cdots)$ does not add additional computational cost to signature generation, but may reduce the cost for signature verification.

To verify whether (r, s) is Alice's signature on m, we recover $k = g^x \bmod p$ from r, s, g, p and y_a and then check whether $hash(k, m)$ is identical to r.

Table 1 shows two shortened versions of DSS, which are denoted by SDSS1 and SDSS2 respectively. Here are a few remarks on the table:

1. the first letter "S" in the name of a scheme stands for "shortened",
2. the parameters p, q and g are the same as those for DSS,
3. x is a random number from $[1, \ldots, q-1]$, x_a is Alice's private key and $y_a = g^{x_a} \bmod p$ is her matching public key,
4. $|t|$ denotes the size or length (in bits) of t,
5. the schemes have the same signature size of $|hash(\cdot)| + |q|$,
6. SDSS1 is slightly more efficient than SDSS2 in signature generation, as the latter involves an extra modular multiplication.

Shortened schemes	Signature (r, s) on a message m	Verification of signature	Length of signature
SDSS1	$r = hash(g^x \bmod p, m)$ $s = x/(r + x_a) \bmod q$	$k = (y_a \cdot g^r)^s \bmod p$ check whether $hash(k, m) = r$	$\lvert hash(\cdot)\rvert + \lvert q\rvert$
SDSS2	$r = hash(g^x \bmod p, m)$ $s = x/(1 + x_a \cdot r) \bmod q$	$k = (g \cdot y_a^r)^s \bmod p$ check whether $hash(k, m) = r$	$\lvert hash(\cdot)\rvert + \lvert q\rvert$

p: a large prime (public to all),
q: a large prime factor of $p - 1$ (public to all),
g: an integer with order q modulo p chosen randomly from $[1, \ldots, p-1]$ (public to all),
$hash$: a one-way hash function (public to all),
x: a number chosen uniformly at random from $[1, \ldots, q-1]$,
x_a: Alice's private key, chosen uniformly at random from $[1, \ldots, q-1]$,
y_a: Alice's public key ($y_a = g^{x_a} \bmod p$).

Table 1: Shortened and Efficient Digital Signature Schemes

2 Advantages

Compared with DSS, SDSS1 and SDSS2 have the following three advantages:

1. Their signatures are shorter: $2|q|$ bits for DSS, while $|hash(\cdot)| + |q|$ bits for SDSS1 and SDSS2. (Typically we have $|q| \approx 2|hash(\cdot)|$.)
2. No modular inversion or division is required in signature verification.
3. They both admit provable security, albeit in the random oracle model.

3 Security Assessment and Consideration

Pointcheval and Stern [46, 47] have proven that a number of digital signature schemes are unforgeable by any adaptive attacker who is allowed to query Alice's signature generation algorithm with messages of his choice [23], in a model where the one-way hash function used in a signature scheme is assumed to behave like a random function (the random oracle model). The core idea behind their unforgeability proofs is based on an observation that such a signature can be viewed as a scheme converted from a 3-move zero-knowledge (ZK) protocol (for proof of knowledge) with respect to a honest verifier. As pointed out by Pointcheval and Stern, with such a signature scheme, unforgeability against a non-adaptive attacker who is not allowed to possess valid message-signature pairs follows from the soundness of the original ZK protocol. Furthermore, as the protocol is zero-knowledge with respect to a honest verifier, the signature scheme converted from it can be efficiently simulated in the random oracle model. This implies that an adaptive attacker is not more powerful than a non-adaptive attacker in the random oracle model.

Turning our attention to SDSS1 and SDSS2, both can be viewed as being converted from a 3-move zero-knowledge protocol (for proof of knowledge) with respect to a honest verifier. Thus Pointcheval and Stern's technique is applicable both to SDSS1 and SDSS2.

Summarizing the above discussions, both SDSS1 and SDSS2 are unforgeable by adaptive attackers, under the assumptions that discrete logarithm is hard (with respect to g chosen uniformly at random) and that the one-way hash function behaves like a random function.

4 Known Limitations

No additional limitations with SDSS1 or SDSS2 are known, when compared with DSS.

5 Patent Issues

The author has not been able to do a patent search, and is not aware of any patent application associated with the shortened signature schemes.

Part II
Signcryption Schemes

6 Description

A signcryption scheme is a cryptographic method that fulfills both the functions of secure encryption and digital signature, but *with a cost smaller than that required by signature-then-encryption.* Using the terminology in cryptography, it consists of a pair of (polynomial time) algorithms (S, U), where S is called the *signcryption algorithm*, while U the *unsigncryption algorithm.* S in general is probabilistic, while U is most likely to be deterministic. (S, U) satisfy the following conditions:

1. *Unique unsigncryptability* — Given a message m of arbitrary length, the algorithm S *signcrypts* m and outputs a *signcrypted text* c. On input c, the algorithm U *unsigncrypts* c and recovers the original message un-ambiguously.

2. *Security* — (S, U) fulfill, simultaneously, the properties of a secure encryption scheme and those of a secure digital signature scheme. These properties mainly include: confidentiality of message contents, unforgeability, and non-repudiation.

3. *Efficiency* — The computational cost, which includes the computational time involved both in signcryption and unsigncryption, and the communication overhead or added redundant bits, of the scheme is *smaller* than that required by the best currently known signature-then-encryption scheme with comparable parameters.

Note that a direct consequence of having to satisfy both the second and third requirements is that "signcryption $\neq$ signature-then-encryption".

Parameters public to all:
p — a large prime
q — a large prime factor of $p-1$
g — an integer with order q modulo p chosen randomly from $[1, \ldots, p-1]$
hash — a one-way hash function whose output has, say, at least 128 bits
KH — a keyed one-way hash function
(E, D) — the encryption and decryption algorithms of a private key cipher
Alice's keys:
x_a — Alice's private key, chosen uniformly at random from $[1, \ldots, q-1]$
y_a — Alice's public key ($y_a = g^{x_a} \bmod p$)
Bob's keys:
x_b — Bob's private key, chosen uniformly at random from $[1, \ldots, q-1]$
y_b — Bob's public key ($y_b = g^{x_b} \bmod p$)

Table 2: Parameters for Signcryption

In describing the signcryption schemes, we will use E and D to denote the encryption and decryption algorithms of a private key cipher such as DES [39]. Encrypting a message m with a key k, typically in the cipher block chaining or CBC mode, is indicated by $E_k(m)$, while decrypting a ciphertext c with k is denoted by $D_k(c)$. In addition we use $KH_k(m)$

to denote hashing a message m with a keyed hash algorithm KH under a key k. An important property of a keyed hash function is that, just like a one-way hash function, it is computationally infeasible to find a pair of messages that are hashed to the same value (or collide with each other). This implies a weaker property that is sufficient for signcryption: given a message m_1, it is computationally intractable to find another message m_2 that collides with m_1. In [2] two methods for constructing a cryptographically strong keyed hash algorithm from a one-way hash algorithm have been demonstrated. For most practical applications, it suffices to define $KH_k(m) = hash(k, m)$, where $hash$ is a one-way hash algorithm.

Assume that Alice has chosen a private key x_a from $[1, \ldots, q-1]$, and made public her matching public key $y_a = g^{x_a} \bmod p$. Similarly, Bob's private key is x_b and his matching public key is $y_b = g^{x_b} \bmod p$. Relevant public and private parameters are summarized in Table 2.

The signcryption and unsigncryption algorithms constructed from a shortened signature are remarkably simple. For Alice to signcrypt a message m to be sent to Bob, she carries out the following operations:

Signcryption of m by Alice the Sender

1. Pick x uniformly at random from $[1, \ldots, q-1]$, and let $k = hash(y_b^x \bmod p)$. Split k into k_1 and k_2 of appropriate length.

2. $r = KH_{k_2}(m, bind\_info)$,
   where *bind_info* contains data that identify Bob the recipient, such as his public key or public key certificate. It may also contain other data items such as Alice's public key.

3. $s = x/(r + x_a) \bmod q$ if SDSS1 is used, or
   $s = x/(1 + x_a \cdot r) \bmod q$ if SDSS2 is used instead.

4. $c = E_{k_1}(m)$.

5. Send to Bob the signcrypted text (c, r, s).

Note that the output of the one-way hash function $hash$ used in defining $k = hash(y_b^x \bmod p)$ should be sufficiently long, say of at least 128 bits, which guarantees that both k_1 and k_2 have at least 64 bits. The main purpose of involving $hash$ in the derivation of k is to simplify our discussions on message confidentiality in Section 3. In practice, k can be defined in a more liberal way, such as $k = y_b^x \bmod p$ and $k = fd(y_b^x \bmod p)$, where fd denotes a folding operation.

The unsigncryption algorithm works by taking advantages of the property that $g^x \bmod p$ can be recovered by Bob from r, s, g, p. On receiving (c, r, s) from Alice, Bob unsigncrypts it as follows:

Unsigncryption of (c, r, s) by Bob the Recipient

1. Recover k from r, s, g, p, y_a and x_b:
   $k = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ if SDSS1 is used, or
   $k = hash((g \cdot y_a^r)^{s \cdot x_b} \bmod p)$ if SDSS2 is used.
2. Split k into k_1 and k_2.
3. $m = D_{k_1}(c)$.
4. accept m as a valid message originated from Alice only if $KH_{k_2}(m, bind\_info)$ is identical to r.

The signcryption scheme that employs the shortened signature scheme SDSS1 is called SCS1, and the signcryption scheme that employs the shortened signature scheme SDSS2 is called SCS2.

The format of the signcrypted text of a message m is depicted in Part (a) of Figure 1. It should be pointed out that in some applications, part of a message m may not need to be encrypted and the creation of the signature part (r, s), especially r, may involve other data in addition to m. A similar observation can be made with the signature-then-encryption approach.

6.1 Working with Signature-Only and Encryption-Only Modes

In practice, some messages may need to be signed only, some encrypted only, while others encrypted partially. For the two digital signcryption schemes SCS1 and SCS2, when a message is sent in clear, they degenerate to signature schemes with direct verifiability by the recipient only. As will be argued in Section 9.2, limiting direct verifiability to the recipient only still preserves non-repudiation, and may represent an advantage for some applications where the mere fact that a message is originated from Alice needs to be kept secret. Furthermore, if Alice uses g instead of Bob's public key y_b in the calculation of k, the schemes becomes corresponding shortened ElGamal based signature schemes with universal verifiability.

In an application where a message requires confidentiality only, one may either switch to a public key encryption scheme such as the ElGamal scheme, or stick to a signcryption scheme. The latter is more efficient than the former, even though authenticity may not be a concern in such an application.

Figure 1: Output Formats of Signcryption and Signature-then-Encryption for a Single Recipient

7 Advantages

The advantage of signcryption over signature-then-encryption lies in the dramatic reduction of computational cost and communication overhead which can be symbolized by the following inequality:

$$\text{Cost(signcryption)} < \text{Cost(signature)} + \text{Cost(encryption)}.$$

With SCS1 and SCS2, this advantage is identifiable in Table 3.

Various schemes	Computational cost	Communication overhead (in bits)
signature-then-encryption based on RSA	EXP=2, HASH=1, ENC=1 (EXP=2, HASH=1, DEC=1)	$\|n_a\| + \|n_b\|$
signature-then-encryption based on "DSS + ElGamal encryption"	EXP=3, MUL=1, DIV=1 ADD=1, HASH=1, ENC=1 (EXP=2.17, MUL=1, DIV=2 ADD=0, HASH=1, DEC=1)	$2\|q\| + \|p\|$
signature-then-encryption based on "Schnorr signature + ElGamal encryption"	EXP=3, MUL=1, DIV=0 ADD=1, HASH=1, ENC=1 (EXP=2.17, MUL=1, DIV=0 ADD=0, HASH=1, DEC=1)	$\|hash(\cdot)\| + \|q\| + \|p\|$
signcryption SCS1	EXP=1, MUL=0, DIV=1 ADD=1, HASH=2, ENC=1 (EXP=1.17, MUL=2, DIV=0 ADD=0, HASH=2, DEC=1)	$\|KH.(\cdot)\| + \|q\|$
signcryption SCS2	EXP=1, MUL=1, DIV=1 ADD=1, HASH=2, ENC=1 (EXP=1.17, MUL=2, DIV=0 ADD=0, HASH=2, DEC=1)	$\|KH.(\cdot)\| + \|q\|$

where
EXP = the number of modular exponentiations (a fractional number indicates an average cost),
MUL = the number of modular multiplications,
DIV = the number of modular division (inversion),
ADD = the number of modular addition or subtraction,
HASH = the number of one-way or keyed hash operations,
ENC = the number of encryptions using a private key cipher,
DEC = the number of decryptions using a private key cipher,
Parameters in the brackets indicate the number of operations involved in "decryption-then-verification" or "unsigncryption".

Table 3: Cost of Signature-Then-Encryption v.s. Cost of Signcryption

What follows is a more detailed analysis of the advantage. The necessity of such an examination is justified by the facts that the computational cost of modular exponentiation is mainly determined by the size of an exponent, and that RSA and discrete logarithm based public key cryptosystems normally employ exponents that are quite different in size. For readers who are not interested in technical details in the comparison, Table 4 summarizes the advantage of SCS1 and SCS2 over discrete logarithm based signature-then-encryption, while Table 5 summarizes that over RSA based signature-then-encryption.

security parameters			saving	saving in
$\lvert p\rvert$	$\lvert q\rvert$	$\lvert KH.(\cdot)\rvert$	average comp. cost	comm. overhead
512	144	72	58%	70.3%
768	152	80	58%	76.8%
1024	160	80	58%	81.0%
1280	168	88	58%	83.3%
1536	176	88	58%	85.3%
1792	184	96	58%	86.5%
2048	192	96	58%	87.7%
2560	208	104	58%	89.1%
3072	224	112	58%	90.1%
4096	256	128	58%	91.0%
5120	288	144	58%	92.0%
8192	320	160	58%	94.0%
10240	320	160	58%	96.0%

$$\text{saving in average comp. cost} = \frac{(5.17-2.17) \text{ modular exponentiations}}{5.17 \text{ modular exponentiations}} = 58\%$$

$$\text{saving in comm. cost} = \frac{|hash(\cdot)|+|q|+|p|-(|KH.(\cdot)|+|q|)}{|hash(\cdot)|+|q|+|p|}$$

Table 4: Saving of Signcryption over Signature-Then-Encryption Using Schnorr Signature and ElGamal Encryption

security parameters			advantage in	advantage in
$\lvert p\rvert(=\lvert n_a\rvert=\lvert n_b\rvert)$	$\lvert q\rvert$	$\lvert KH.(\cdot)\rvert$	average comp. cost	comm. overhead
512	144	72	0%	78.9%
768	152	80	14.2%	84.9%
1024	160	80	32.3%	88.3%
1280	168	88	43.1%	90.0%
1536	176	88	50.3%	91.4%
1792	184	96	55.5%	93.1%
2048	192	96	59.4%	93.0%
2560	208	104	64.8%	94.0%
3072	224	112	68.4%	94.5%
4096	256	128	72.9%	95.0%
5120	288	144	75.6%	96.0%
8192	320	160	83.1%	97.0%
10240	320	160	86.5%	98.0%

$$\text{advantage in average comp. cost} = \frac{0.375(|n_a|+|n_b|)-3.25|q|}{0.375(|n_a|+|n_b|)}$$

$$\text{advantage in comm. cost} = \frac{|n_a|+|n_b|-(|KH.(\cdot)|+|q|)}{|n_a|+|n_b|}$$

Table 5: Advantage of Signcryption over RSA based Signature-Then-Encryption with *Small Public Exponents*

7.1 A Comparison with Signature-Then-Encryption Using Schnorr Signature and ElGamal Encryption

7.1.1 Saving in computational cost

With the signature-then-encryption based on Schnorr signature and ElGamal encryption, the number of modular exponentiations is three, both for the process of signature-then-encryption and that of decryption-then-verification.

Among the three modular exponentiations for decryption-then-verification, two are used in verifying Schnorr signature. More specifically, these two exponentiations are spent in computing $g^s \cdot y_a^r \bmod p$. Using a technique for fast computation of the product of multiple exponentials with the same modulo which has been attributed to Shamir (see Appendix B), $g^s \cdot y_a^r \bmod p$ can be computed, on average, in $(1 + 3/4)|q|$ modular multiplications. Since a modular exponentiation can be completed, on average, in about $1.5|q|$ modular multiplications when using the classical "square-and-multiply" method, $(1 + 3/4)|q|$ modular multiplications is computationally equivalent to 1.17 modular exponentiations. Thus with "square-and-multiply" and Shamir's technique, the number of modular exponentiations involved in decryption-then-verification can be reduced from 3 to 2.17. The same reduction techniques, however, cannot be applied to the sender's computation. Consequently, the combined computational cost of the sender and the recipient is 5.17 modular exponentiations.

In contrast, with SCS1 and SCS2, the number of modular exponentiations is one for the process of signcryption and two for that of unsigncryption respectively. Since Shamir's technique can also be used in unsigncryption, the computational cost of unsigncryption is about 1.17 modular exponentiations. The total average computational cost for signcryption is therefore 2.17 modular exponentiations. This represents a

$$\frac{5.17 - 2.17}{5.17} = 58\%$$

reduction in average computational cost.

7.1.2 Saving in communication overhead

The communication overhead measured in bits is $|hash(\cdot)| + |q| + |p|$ for the signature-then-encryption based on Schnorr signature and ElGamal encryption, and $|KH.(\cdot)| + |q|$ for the two signcryption schemes SCS1 and SCS2, where $|x|$ refers to the size of a binary string, KH is a keyed hash function and $hash$ is a one-way hash function (used in Schnorr signature, but not the one used in signcryption). Hence the saving in communication overhead is

$$\frac{|hash(\cdot)| + |q| + |p| - (|KH.(\cdot)| + |q|)}{|hash(\cdot)| + |q| + |p|}$$

Assuming that the one-way hash function $hash$ used in the signature-then-encryption scheme and the keyed hash function KH used in the signcryption scheme share the same output length, the reduction in communication overhead is $|p|$. For the minimum security parameters recommended for use in current practice: $|KH.(\cdot)| = |hash(\cdot)| = 72$, $|q| = 144$ and $|p| = 512$, the numerical value for the saving is 70.3%. One can see that the longer the prime p, the larger the saving.

The above discussions can be summarized as:

$$\text{Cost(signcryption)} \approx \text{Cost(shortened signature)}$$

for the two signcryption schemes SCS1 and SCS2.

7.2 A Comparison with Signature-Then-Encryption Using RSA

7.2.1 Advantage in computational cost

With RSA, it is a common practice to employ a relatively small public exponent e for encryption or signature verification, although cautions should be taken in light of recent progress in cryptanalysis against RSA with an small exponent (see for example [17, 16]). Therefore the main computational cost is in decryption or signature generation which generally involves a modular exponentiation with a *full size* exponent d, which takes on average 1.5ℓ modular multiplications using the "square-and-multiply" method, where ℓ indicates the size of the RSA composite involved. With the help of the Chinese Remainder Theorem, the computational expense for RSA decryption can be reduced, theoretically, to a quarter of the expense with a full size exponent, although in practice it is more realistic to expect the factor to be between 1/4 and 1/3. To simplify our discussion, we assume that the maximum speedup is achievable, namely the average computational cost for RSA decryption is $\frac{1.5}{4}\ell = 0.375\ell$ modular multiplications.

For the signature-then-encryption based on RSA, four (4) modular exponentiations are required (two with public exponents and the other two with private exponents). Assuming small public exponents are employed, the computational cost will be dominated by the two modular exponentiations with full size private exponents. When the Chinese Remainder Theorem is used, this cost is on average $0.375(|n_a| + |n_b|)$ modular multiplications, where n_a and n_b are the RSA composites generated by Alice and Bob respectively.

As discussed earlier, the two signcryption schemes SCS1 and SCS2 both involve, on average, 2.17 modular exponentiations, or equivalently $3.25|q|$ modular multiplications, assuming the "square-and-multiply" method and Shamir's technique for fast computation of the product of exponentials are used. This shows that the signcryption schemes represent an advantage of

$$\frac{0.375(|n_a| + |n_b|) - 3.25|q|}{0.375(|n_a| + |n_b|)}$$

in average computational cost over the RSA based signature-then-encryption. For small security parameters, the advantage is less significant. This situation, however, changes dramatically for large security parameters: consider $|n_a| = |n_b| = |p| = 1536$ and $|q| = 176$ which are recommended to be used for long term (say more than 20 years) security, the signcryption schemes show a 50.3% saving in computation, when compared with the signature-then-encryption based on RSA.

The advantage of the signcryption schemes in computational cost will be more visible, should large public exponents be used in RSA.

7.2.2 Advantage in communication overhead

The signature-then-encryption based on RSA expands each message by a factor of $|n_a|+|n_b|$ bits, which is multiple times as large as the communication overhead $|KH.(\cdot)| + |q|$ of the two signcryption schemes SCS1 and SCS2. Numerically, the advantage or saving of the signcryption schemes in communication overhead over the signature-then-encryption based on RSA is as follows:

$$\frac{|n_a| + |n_b| - (|KH.(\cdot)| + |q|)}{|n_a| + |n_b|}$$

For $|n_a| = |n_b| = 1536$, $|q| = 176$ and $|KH.(\cdot)| = 88$, the advantage is 91.4%. The longer the composites n_a and n_b, the larger the saving by signcryption.

Note that we have chosen not to compare the signcryption schemes with unbalanced RSA recently proposed by Shamir [51]. The main reason is that while the new variant of RSA is attractive in terms of its computational efficiency, its security has yet to be further scrutinized by the research community.

7.3 Remarks on the Comparison

The above comparison between signcryption and signature-then-encryption should only be interpreted as a rough indicator for the relative efficiency of the two different paradigms. The comparison has been based on the assumption that only basic modular exponentiation techniques are used, these being the "square-and-multiply" method, Shamir's method for fast evaluation of the product of several exponentials with the same modulo, and in the case of RSA, the Chinese Remainder Theorem.

Instead of "square-and-multiply", other fast exponentiation methods such as sliding-window exponentiation may be used. For the RSA cryptosystem, signature generation and decryption can be sped up by adopting fast algorithms for fixed-exponent exponentiation. A notable example of such algorithms is addition chain exponentiation. On the other hand, discrete logarithm based cryptosystems, including the signcryption schemes, can be made more efficient using a number of strategies. Examples of these strategies are (1) elliptic curves, (2) Lenstra's new sub-groups based on cyclotomic polynomials [34], and (3) fast algorithms for fixed-base exponentiation. When all these techniques are used, the resultant comparative indicator for the relative efficiency of signcryption and signature-then-encryption may oscillate around the values shown above.

A final remark is that our comparison has been carried out in terms of the *absolute* number of bits and computational operations that can be saved by signcryption. Comparisons can also be made in terms of savings *relative* to the size of an entire data packet which may include a (scrambled) message, the identifiers of a sender and a recipient, signatures, public key certificates, time-stamps, and so on. A problem with relative comparisons is that their indicators decrease when the size of a message increases, which may render obscure the significant advantages of signcryption over signature-then-encryption.

7.4 How the Parameters are Chosen

Advances in fast computers help an attacker in increasing his capability to break a cryptosystem. To compensate this, larger security parameters, including $|n_a|$, $|n_b|$, $|p|$, $|q|$ and $|KH.(\cdot)|$ must be used in the future. From an analysis by Odlyzko [43] on the hardness of discrete logarithm, one can see that unless there is an algorithmic breakthrough in solving the factorization or discrete logarithm problem, $|q|$ and $|KH.(\cdot)|$ can be increased at a smaller pace than can $|n_a|$, $|n_b|$ and $|p|$. Thus, as shown in Tables 4 and 5, the saving or advantage in computational cost and communication overhead by signcryption will be more significant in the future when larger parameters must be used.

The selection of security parameters $|p|$, $|q|$, $|n_a|$ and $|n_a|$ in Tables 4 and 5, has been partially based on recommendations made in [43]. The parameter values in the tables, however, are indicative only, and can be determined flexibly in practice. We also note that choosing $|KH.(\cdot)| \approx |q|/2$ is due to the fact that using Shank's baby-step-giant-step or Pollard's rho method, the complexity of computing discrete logarithms in a sub-group of order q is $O(\sqrt{q})$ (see [35]). Hence choosing $|KH.(\cdot)| \approx |q|/2$ will minimize the communication overhead of the signcryption schemes SCS1 and SCS2. Alternatively, one may decide to choose $KH.(\cdot) \in [1, \ldots, q-1]$ which can be achieved by setting $|KH.(\cdot)| = |q| - 1$. This will not affect the computational advantage of the signcryption schemes, but slightly increase their communication overhead.

7.5 Other Advantages

7.5.1 Past Recovery

Consider the following scenario: Alice signs and encrypts a message and sends it to Bob. A while later, she finds that she wants to use the contents of the message again.

To satisfy Alice's requirement, her electronic mail system has to store some data related to the message sent. And depending on cryptographic algorithms used, Alice's electronic mail system may either (1) keep a copy of the signed and encrypted message as evidence of transmission, or (2) in addition to the above copy, keep a copy of the original message, either in clear or encrypted form.

A cryptographic algorithm or protocol is said to provide a past recovery ability if Alice can recover the message from the signed and encrypted message alone using her private key. While both signcryption and "signature-then-encryption-with-a-static-key" provide past recovery, "signature-then-encryption" does not.

One may view "signature-then-encryption" as an information "black hole" with respect to Alice the sender: whatsoever Alice drops in the "black hole" will never be retrieval to her, unless a separate copy is properly kept. Therefore signcryption schemes are more economical with regard to secure and authenticated transport of large data files. It is even more so when Alice has to broadcast the same message to a large number of recipients.

Obviously a cryptographic algorithm or protocol provides past recovery if and only if it does *not* provide forward secrecy with respect to Alice the sender's long term private key.

8 Security Assessment and Consideration

Like any cryptosystem, security of signcryption in general has to address two aspects: (1) to protect what, and (2) against whom. With the first aspect, we wish to prevent the contents of a signcrypted message from being disclosed to a third party other than Alice, the sender, and Bob, the recipient. At the same time, we also wish to prevent Alice, the sender, from being masquerade by other parties, including Bob. With the second aspect, we consider the most powerful attackers one would be able to imagine in practice, namely adaptive attackers who are allowed to have access to Alice's signcryption algorithm and Bob's unsigncryption algorithm.

We say that a signcryption scheme is secure if the following conditions are satisfied:

1. Unforgeability — it is computationally infeasible for an adaptive attacker (who may be a dishonest Bob) to masquerade Alice in creating a signcrypted text.

2. Non-repudiation — it is computationally feasible for a third party to settle a dispute between Alice and Bob in an event where Alice denies the fact that she is the originator of a signcrypted text with Bob as its recipient.

3. Confidentiality — it is computationally infeasible for an adaptive attacker (who may be any party other than Alice and Bob) to gain any partial information on the contents of a signcrypted text.

The following sub-sections are devoted to the security argument of the signcryption schemes SCS1 and SCS2.

8.1 Unforgeability

Regarding forging Alice's signcryption, a dishonest Bob is in the best position to do so, as he is the only person who knows x_b which is required to directly verify a signcrypted text from Alice. In other words, the dishonest Bob is the most powerful attacker we should look at. Given the signcrypted text (c, r, s) of a message m from Alice, Bob can use his private key x_b to decrypt c and obtain $m = D_{k_2}(c)$. Thus the original problem is reduced to one in which Bob is in possession of (m, r, s). The latter is identical to the unforgeability of SDSS1 or SDSS2.

As argued in Section 3, SDSS1 and SDSS2 are unforgeable. Therefore we conclude that both signcryption schemes SCS1 and SCS2 are unforgeable against adaptive attacks, under the assumption that the keyed hash function behaves like a random function.

8.2 Non-repudiation

Signcryption requires a repudiation settlement procedure different from the one for a digital signature scheme is required. In particular, the judge would need Bob's cooperation in order to correctly decide the origin of the message. In what follows we describe four possible repudiation settlement procedures, each requiring a different level of trust on the judge's side.

8.2.1 With a Trusted Tamper-Resistant Device

If a tamper-resistant device is available, a trivial settlement procedure starts with the judge asking Bob to provide the device with q, p, g, y_a, y_b, m, c, r, s and his private key x_b, together with certificates for y_a and y_b. The tamper-resistant device would follow essentially the same steps used by Bob in unsigncrypting (c, r, s). It would output "yes" if it can recover m from (c, r, s), and "no" otherwise. The judge would then take the output of the tamper-resistant device as her decision. Note that in this case, Bob puts his trust completely on the device, rather than on the judge.

8.2.2 By a Trusted Judge

If the judge is trusted, achieving correct repudiation settlement by the judge is again trivial: Bob simply presents to the the judge x_b together with other data items. Note that both in this case and in the case of a tamper-resistant device, Bob's presenting k, rather than x_b, to the judge or the device without convincing her or the device that k satisfies the condition of $k = hash(u^{x_b} \bmod p)$, would open up a door for a dishonest Bob to frame Alice by providing a message made up by himself, where x_b is Bob's private key, $u = (y_a \cdot g^r)^s \bmod p$ for SCS1, and $u = (g \cdot y_a^r)^s \bmod p$ for SCS2.

Note that once being given x_b, the judge can do everything Bob can with x_b.

8.2.3 By a Less Trusted Judge

Another possible solution would be for Bob to present $v = u^{x_b} \bmod p$, rather than x_b, to the judge. Bob and the judge then engage in a zero-knowledge interactive/non-interactive proof/argument protocol (with Bob as a prover and the judge as a verifier), so that Bob can convince the judge of the fact that v does have the right form. (A possible candidate protocol is a 4-move zero-knowledge proof protocol developed in [12].)

Bob has to be aware of the fact that with this repudiation settlement procedure, the judge can obtain from v, r, s and y_b the Diffie-Hellman shared key between Alice and Bob, namely $k_{DH,ab} = g^{x_a x_b} \bmod p$ $(= v^{1/s} y_b^{-r} \bmod p$ for SCS1). With $k_{DH,ab}$, the judge can find out v^* for other communication sessions between Alice and Bob, and hence recover the corresponding messages ($v^* = k_{DH,ab}^{s^*} y_b^{r^* \cdot s^*} \bmod p$ for SCS1). Therefore Bob may not rely on this repudiation settlement procedure if the judge is not trusted by either Alice or Bob.

8.2.4 By any (Trusted/Untrusted) Judge

Now we describe a repudiation settlement procedure that works even in the case when the judge corrupts and is not trusted. The procedure uses techniques in zero-knowledge proofs/arguments [1] and guarantees that the judge can make a correct decision, with no useful information on Bob's private key x_b being leaked out to the judge.

First Bob presents following data to the judge: q, p, g, y_a, y_b, m, c, r, s and certificates for y_a and y_b. Note that Bob does not hand out x_b, k or $v = u^{x_b} \bmod p$. The judge then verifies the authenticity of y_a and y_b. If satisfied both with y_a and y_b, the judge computes $u = (y_a \cdot g^r)^s \bmod p$ when SCS1 is used, and $u = (g \cdot y_a^r)^s \bmod p$ when SCS2 is used instead. Bob and the judge then engage in a zero-knowledge interactive protocol, with Bob as a prover and the judge as a verifier.

The goal of the protocol is for Bob to convince the judge of the fact that he knows a satisfying assignment $z = x_b$ to the following Boolean formula φ:

$$\begin{aligned} \varphi(z) \quad = \quad & (g^z \bmod p == y_b) \wedge \\ & (D_{k_1}(c) == m) \wedge \\ & (KH_{k_2}(D_{k_1}(c)) == r) \end{aligned}$$

where k_1 and k_2 are defined by $(k_1, k_2) = hash(u^z \bmod p)$, and $==$ denotes equality testing.

φ is clearly a satisfiable Boolean formula in the class of NP. There are a large number of zero-knowledge proof/argument protocols for NP statements in the literature. Some zero-knowledge protocols are based on assumptions on specific computational problems such integer factorization and discrete logarithm, while others work with general complexity assumptions such as the existence of one-way functions. Here we only mention one of such protocols recently proposed in [3]. This zero-knowledge argument protocol assumes that both Bob (the prover) and the judge (the verifier) are polynomially bounded in computing time, which matches our cryptographic setting. In other words, it is a zero-knowledge argument protocol. It consists of only four moves of messages between Bob and the judge, and any one-way function whose input and out are of equal length can be used to build a so-called bit-commitment scheme used in the protocol. In practice, the one-way function can be instantiated with a one-way hash function or a secure block cipher. After an execution of the protocol, the judge announces that (c, r, s) is originated from Alice to Bob only if she is convinced that the Boolean formula φ is satisfiable.

[1] The main difference between a proof and an argument in the context of zero-knowledge protocols is that, while an argument assumes that a prover runs in polynomial time, a proof works even if a prover has unlimited computational power. A zero-knowledge argument suffices for most cryptographic applications, including repudiation settlement in signcryption.

To summarize the above discussion on interactions between Bob and the judge, one (such as an implementation of a signcryption scheme) would first correctly figure out the Boolean formula φ. Then one would select a suitable one-way function. And finally one would code the four-move protocol specified in [3].

Properties of the protocol include: (1) the judge always correctly announces that (c, r, s) is originated from Alice when it is indeed so; (2) the probability is negligibly small for the judge to declare that (c, r, s) is originated from Alice when in fact it is not; (3) no useful information on Bob's private key x_b is leaked to the judge (or any other parties).

Two remarks on the interactive repudiation settlement procedure follow. First, the message m may be dropped from the data items handed over to the judge, if Bob does not wish to reveal the contents of m to the judge. Second, Bob may include k into the data handed over to the judge if k is defined as $k = hash(y_b^x \bmod p)$ in which a one-way hash function $hash$ is involved. This will reduce the computation and communication load involved in the interactions without compromising the security of x_b, especially when $hash$ is a cryptographically strong function that does not leak information on its input.

Finally we note that if Bob and the judge share a common random bit string, then the number of moves of messages between Bob and the judge can be minimized to 1, by the use of a non-interactive zero-knowledge proof protocol such as the one proposed in [32].

8.3 Confidentiality

Finally we consider the confidentiality of message contents. We use SCS1 as an example, as discussions for SCS2 are similar. Given the signcrypted text (c, r, s) of a message m from Alice, an attacker can obtain $u = (y_a \cdot g^r)^s = g^x \bmod p$. Thus to the attacker, data related to the signcrypted text of m include: q, p, g, $y_a = g^{x_a} \bmod p$, $y_b = g^{x_b} \bmod p$, $u = g^x \bmod p$, $c = E_{k_1}(m)$, $r = KH_{k_2}(m)$, and $s = x/(r + x_a) \bmod q$.

We wish to show that it is computationally infeasible for the attacker to find out any partial information on the message m from the related data listed above. We will achieve our goal by reduction: we will reduce the confidentiality of another encryption scheme to be defined shortly (called C_{kh} for convenience) to the confidentiality of SCS1.

The encryption scheme C_{kh} is based on ElGamal encryption scheme. With this encryption scheme, the ciphertext of a message m to be sent to Bob is defined as ($c = E_{k_1}(m)$, $u = g^x \bmod p$, $r = KH_{k_2}(m)$) where (1) x is chosen uniformly at random from $[1, \ldots, q-1]$, and (2) $(k_1, k_2) = k = hash(y_b^x \bmod p)$, It turns out C_{kh} is a slightly modified version of a scheme that has received special attention in [53, 5]. (See also earlier work [59].) Using a similar argument as that in [53, 5], we can show in the following that for C_{kh}, it is computationally infeasible for an adaptive attacker to gain any partial information on m.

We assume that there is an attacker for SCS1. Call this attacker A_{SCS1}. We show how A_{SCS1} can be translated into one for C_{kh}, called $A_{C_{kh}}$. Recall that for a message m, the input to A_{SCS1} includes q, p, g, $y_a = g^{x_a} \bmod p$, $y_b = g^{x_b} \bmod p$, $u = g^x \bmod p$, $c = E_{k_1}(m)$, $r = KH_{k_2}(m)$. With the attacker $A_{C_{kh}}$ for C_{kh}, however, its input includes: q, p, g, $y_b = g^{x_b} \bmod p$, $u = g^x \bmod p$, $c = E_{k_1}(m)$, and $r = KH_{k_2}(m)$. One immediately identifies that two numbers that correspond to y_a and s which are needed by A_{SCS1} as part of its input are currently missing from the input to $A_{C_{kh}}$. Thus, in order for $A_{C_{kh}}$ to "call" the attacker A_{SCS1} "as a sub-routine", $A_{C_{kh}}$ has to create two numbers corresponding to y_a and s in the input to A_{SCS1}. Call these two yet-to-be-created numbers y'_a and s'. y'_a and s' have to have the right form so that $A_{C_{kh}}$ can "fool" A_{SCS1}. It turns out that such y'_a and s' can be easily created by $A_{C_{kh}}$ as follows: (1) pick a random number s' from $[1, \ldots, q-1]$.

(2) let $y'_a = u^{1/s'} \cdot g^{-r} \bmod p$.

From the reduction, we see that if there is an attacker that finds any partial information on a message in SCS1, then the attacker can be used to find partial information on a message in C_{kh}. As we have already proven the confidentiality of C_{kh}, we conclude that no attacker for SCS1 can find any partial information on a message.

There are two minor technicalities that remain to be removed, before we can complete our full formal proof of the confidentiality of SCS1. First we have assumed that A_{SCS1} works for all y_a and y_b. Namely A_{SCS1} is universally powerful with respect to y_a and y_b. This requirement on A_{SCS1} can be relaxed to that it works for a non-negligible portion of all valid public keys. Second, our discussions on the proof have been informal. This problem can be removed by the use of a standard model for security established in [22], in conjunction with proof techniques used in [59] or [53, 5].

9 Known Limitations

9.1 Forward Secrecy

A cryptographic primitive or protocol provides forward secrecy with respect to a long term private key if compromise of the private key does not result in compromise of security of previously communicated or stored messages.

With "signature-then-encryption", since different keys are involved in signature generation and public key encryption, forward secrecy is in general guaranteed with respect to Alice's long term private key. (Nevertheless, loss of Alice's private key renders her signature forgeable.) In contrast, with the signcryption schemes, it is easy to see that knowing Alice's private key alone is sufficient to recover the original message of a signcrypted text. Thus no forward secrecy is provided by the signcryption schemes with respect to Alice's private key. A similar observation applies to "signature-then-encryption-with-a-static-key" with respect to Alice's shared static key.

Forward secrecy has been regarded particularly important for session key establishment [20]. However, to fully understand its implications to practical security solutions, we should identify (1) how one's long term private key may be compromised, (2) how often it may happen, and (3) what can be done to reduce the risks of a long key being compromised. In addition, the cost involved in achieving forward secrecy is also an important factor that should be taken into consideration.

There are mainly three causes for a long term private key being compromised: (1) the underlying computational problems are broken; (2) a user accidentally loses the key; (3) an attacker breaks into the physical or logical location where the key is stored.

As a public key cryptosystem always relies on the (assumed) difficulty of certain computational problems, breaking the underlying problems renders the system insecure and useless. Assuming that solving underlying computational problems is infeasible, an attacker would most likely try to steal a user's long term key through such a means as physical break-in.

To reduce the impact of signcryption schemes' lack of forward secrecy on certain security applications, one may suggest users change their long term private keys regularly. In addition, a user may also use techniques in secret sharing [50] to split a long term private key into a number of shares, and keep each share in a separate logical or physical location. This would significantly reduce the risk of a long term key being compromised, as an attacker now faces a difficult task to penetrate in a larger-than-a-threshold number of locations in a limited period of time.

9.2 Repudiation Settlement

Now we turn to the problem of how to handle repudiation. With signature-then-encryption, if Alice denies the fact that she is the originator of a message, all Bob has to do is to decrypt the ciphertext and present to a judge (say Julie) the message together with its associated signature by Alice, based on which the judge will be able to settle a dispute.

With digital signcryption, however, the verifiability of a signcryption is in normal situations limited to Bob the recipient, as his private key is required for unsigncryption. Now consider a situation where Alice attempts to deny the fact that she has signcrypted and sent to Bob a message m. Similarly to signature-then-encryption, Bob would first unsigncrypt the signcrypted text, and then present the following data items to a judge (Julie): q, p, g, y_a, y_b, m, r, and s. One can immediately see that the judge cannot make a decision using these data alone. To solve this problem, Bob and the judge have to engage in an interactive zero-knowledge proof/argument protocol. Details will be discussed in Section 8.2.

At the first sight, the need for an interactive repudiation settlement procedure between Bob and the judge may be seen as a drawback of signcryption. Here we argue that interactive repudiation settlement will not pose any problem in practice and hence should not be an obstacle to practical applications of signcryption. In the real life, a message sent to Bob in a secure and authenticated way is meant to be readable by Bob only. Thus if there is no dispute between Alice and Bob, direct verifiability by Bob only is precisely what the two users want. In other words, in normal situations where no disputes between Alice and Bob occur, the full power of universal verifiability provided by digital signature is never needed. (For a similar reason, traditionally one uses signature-then-encryption, rather than encryption-then-signature. See also [13] for potential risks of forgeability accompanying encryption-then-signature.) In a situation where repudiation does occur, interactions between Bob and a judge would follow. This is very similar to a dispute on repudiation in the real world, say between a complainant (Bob) and a defendant (Alice), where the process for a judge to resolve the dispute requires in general interactions between the judge and the complainant, and furthermore between the judge and an expert in hand-written signature identification, as the former may rely on advice from the latter in correctly deciding the origin of a message. The interactions among the judge, Bob the recipient and the expert in hand-written signature identification could be time-consuming and also costly.

9.3 "Community" v.s. World Orientation

With the signcryption schemes, both Alice and Bob have to use the same p and g. So they basically belong to the same "community" defined by p and g. Such a restriction does not apply to "signature-then-encryption".

Similar restrictions apply to "signature-then-encryption-with-a-static-key" where the static key is derived from the Diffie-Hellman key $g^{x_a x_b} \bmod p$, or a key pre-distribution scheme [36]. Such restrictions seem to be inherent with cryptographic protocols based on the Diffie-Hellman public key cryptosystem [19]. A recent example of such protocols is an Internet key agreement protocol based on ISAKMP and Oakley [25].

In the case where a static key is a pre-shared random string between Alice and Bob, whether or not Alice and Bob belong to the same "community" will be determined by the underlying protocol for distributing the pre-shared random string.

In theory, the requirement that both Alice and Bob belong to the same "community" does limit the number of users with whom Alice can communicate using a signcryption scheme. In reality, however, all users belong to several "communities", and they tend to communicate more with users in the same group than with outsiders: users (including banks and individuals) of a certain type of digital cash payment system, employees of a company and citizens of a country, to name a few. Therefore the "community" oriented nature of signcryption schemes may not bring much inconvenience to their use in practice.

10 Patent Issues

Monash University has lodged applications for patents both in Australia and USA. "Reasonable and non-discriminatory" patent licensing is available from Monash.

Part III
Unforgeable and Non-repudiateable Key Agreement Schemes

11 Introduction

There has been an extremely large body of research in the area of key establishment since the publication of the landmark paper by Diffie and Hellman [19], which has resulted in a situation where one may find numerous protocols in the literature, each of which may have different properties. A primary reason behind the emergence of such a large number of key establishment protocols can perhaps be attributed to the many different dimensions of key establishment. The rest of this section is intended as a brief summary of the many facets in key establishment, with a view to properly position this work in the research area. The reader who wishes to find more detailed information on various key establishment methods proposed so far may consult Chapter 12 of [38] and references listed in the handbook.

11.1 Security

A session key established by an execution of a protocol should be known only to the two participants involved, and also to a KDC or key distribution center if the protocol involves the KDC. Security of the session key should not be compromised under all the possible attacks that might be encountered in a particular environment where the protocol will be employed. Typical attacks include (1) inferring a session key via (passive) eavesdropping, (2) replaying past messages, (3) interleaving messages from one protocol execution with another, (4) deducing a session key with a known past session key.

To defend against replay attack, a session key has to be fresh and new. Freshness is usually achieved through the use of a time-varying quantity, such as a time-stamp or a nonce (random number) or a combination of both. The main difference between the use of a time-stamp and a nonce is that the former assumes the existence of a synchronized clock between two participants (and/or between participants and a KDC in a KDC-based protocol). While the requirement of a synchronized clock in a local area network should be readily fulfilled, it may pose a problem in a wide area network.

Interleaving attacks were extensively studied in [7]. In an attempt to formalize the notion of resisting against replay and interleaving attacks, a concept of “matching of protocol histories” was also introduced in [7]. The concept was extended to the notion of “matching protocol runs” in [20]. The latter was further refined to a general notion in [4] where the authors presented the first formal security proofs for two authenticated key establishment protocols. In the complexity-theoretic model proposed in [4], an attacker has full control over all the transactions among participants. In particular, the attack is allowed to engage himself in as many executions of a key establishment protocol as he wishes at any particular point of time. Some recent progress in this line of research can be found in [9, 26].

In addition to the above attacks, some applications may require that a key establishment protocol have the capacity of minimizing the impact of compromise of a long term secret key. This requirement is called “(perfect) forward secrecy”. A key establishment protocol is said to offer forward secrecy with respect to a particular participant if compromise of the participant’s long term secret key does not result in the exposure of past session keys.

Clearly, a secret key based protocol cannot offer forward secrecy with respect to a participant or a KDC whose long term secret key is involved in encrypting key materials. The same can be said with a public key based protocol, with respect to a participant whose long term private key is used to extract key materials. If a participant's private key is only involved in the creation of a digital signature on messages, compromise of the private key may not directly expose a session key, rather it may result in impersonation of the participant by an attacker. The Internet Oakley key determination protocol [44, 25] has been designed to achieve forward secrecy. The reader is referred to Section 15 for a detailed discussion on more economical techniques for reducing the chance that a long term secret key might be compromised.

Another relevant issue is on formal proofs or arguments of security. Protocols that have been designed to resist against known attacks and have survived these attacks are said to admit heuristic security. Since such protocols are not proved to be secure against *all* attacks that may arise in practice, many of them have been found to be flawed [11]. Therefore ideally one would like to have protocols that admit *provable security* against all attacks. A complexity-theoretic approach towards provable security was initiated in [4] where the provable security of two key establishment protocols based on secret key cryptosystems was also presented.

11.2 Authentication v.s. Identification

Entity authentication is a process by which a participant is convinced of the identity of another participant. Entity authentication can be unilateral (one-way) or mutual (two-way). In a mutual authentication protocol, both participants wish to be convinced that the other participant is indeed who he/she claims to be.

A concept that is closely related to and often confused with entity authentication is *identification*. While the aim of identification is similar to entity authentication, namely for one participant, say Alice, to convince another participant, say Bob, of her identity, identification satisfies a more stringent requirement: no participant other than Alice can prove that he or she is Alice, even to him or herself. The difference between entity authentication and identification is made clear by examining a protocol based on a shared static key between Alice and Bob. Alice and Bob can mutually authenticate each other using the static key in three moves [30]. However, such a protocol is not an identification protocol, since whatever produced by Alice using the shared key can also be created by Bob, and vice versa.

An implicit requirement of key establishment in practical applications is that it offers at least unilateral entity authentication or identification. A protocol that offers both key establishment and (unilateral/mutual) entity authentication or identification is called a (unilaterally/mutually) authenticated key establishment.

In secret key cryptosystem based protocols, entity authentication is almost exclusively achieved by way of challenge-and-response. A key establishment protocol employing public key cryptosystems can offer identification by the use of digital signature.

Technically, some entity authentication or identification protocols may be modified to carry session keys, and conversely, an authenticated key establishment can also be used for the purpose of entity authentication or identification.

Finally there is another concept related to authentication. That is key confirmation whose purpose is for a participant to acknowledge that he or she is indeed in possession of a session key. The last message of a key establishment protocol can be optionally composed of an authentication tag. The tag may be generated from a newly agreed session key by $tag = MAC_{key}(known\_message)$, where MAC may be instantiated with either a message authentication code or a keyed hash function.

11.3 Unforgeability and Non-repudiation

In some applications, a participant may require that his or her messages cannot be forged by other participants. Symmetrically, the recipient of a message, especially of one that contains key materials, may require that the sender of the message cannot repudiate at a later stage the fact that he or she is the originator of the message. We envisage that in electronic commerce, non-repudiation and unforgeability of key materials and actual communication sessions that employ a key derived from the key materials is of particular importance.

Both unforgeability and non-repudiation can be achieved by using digital signature. With a secret key based protocol, however, neither unforgeability nor non-repudiation can be achieved, unless the protocol involves a KDC, possibly together with tamper-resistant devices.

Unforgeability and non-repudiation are closely related to identification. In particular, unforgeability or non-repudiation cannot be accomplished unless a key establishment protocol is also an identification protocol.

Let us turn our attention to high speed networks. In a (draft) document prepared by the ATM Forum [15], *accountability* for all ATM network service invocations and management activities, as well as for each individual entity's actions, has been identified as one of the four main security objectives, with the other three being confidentiality, data integrity and availability. This objective would not be achieved unless key establishment protocols employed fulfill unforgeability and non-repudiation.

11.4 Transport v.s. Exchange

We distinguish between two types of key establishment protocols: *key (material) exchange* protocols [2] and *key (material) transport* protocols. With a key exchange protocol, a shared session key is derived from joint key materials from both participants. Such a protocol requires both participants involved to exchange key materials. In contrast, with a key transport protocol, key materials from which a session key is derived are created by one participant and transferred to the other. A key exchange protocol may be preferred to a key transport protocol in certain applications where a session key is required to be "fair", in that it is dependent on both participants' key materials. However, one should distinguish between key material exchange and shared generation of random numbers as achieved in threshold cryptography [18]. In particular, with a key exchange protocol a participant who is in a position to see, prior to producing his key materials, those from the other participant may control the resultant session key by carefully choosing his key materials. In this sense, a key (material) exchange protocol is essentially the same as a key (material) transport protocol. In general, truly "fair" session key generation cannot be achieved without the involvement of computationally expensive bit/sequence commitment, and hence it is our view that "fairness" should not be set as a goal of key establishment.

Some examples of key transport protocols are Kerberos [33] and X.509 strong authenticated protocols [29]. The most prominent representative of key exchange protocols is the Diffie-Hellman protocol [19].

[2]The terms of *key exchange protocol* and *key agreement protocol* can be used interchangeably.

11.5 Secret v.s. Public Key Cryptosystems

Prior to the execution of a key establishment protocol, two participants may or may not have shared static keys in their hands. In the case of having a shared static key, the most efficient way for them to establish a fresh session key is to use a key establishment protocol built on a secret key (or symmetric) cryptosystem. Both Kerberos [33] and KryptoKnight [8, 30] are based on secret key cryptosystems.

A shared static key may have been established in three different ways: (1) an explicit key established previously, (2) an implicit key defined in a key pre-distribution scheme [36], and (3) an implicit key defined in the Diffie-Hellman protocol [19].

On the other hand, if the two participants do not have a shared static key, they may have to use a public key cryptosystem which is not as efficient as a secret key cryptosystem, unless they can ask for help from a key distribution center with whom both participants have a separately shared static key. Examples of key establishment using public key cryptosystems are X.509 strong authenticated key transport protocols [29], Beller-Yacobi protocols [6] and the Internet Oakley key determination protocol [44, 25]. The latter is a modified version of a station-to-station protocol proposed in [20].

Note that for efficiency reason, a public key based protocol usually uses secret key cryptosystems in encrypting data. Other cryptographic primitives such as authentication codes and one-way or keyed hashing may also be used in combination with a secret or public key based protocol.

11.6 Trusted Third Party

In some applications, there may be a trusted third party called a key distribution center (KDC), with which each participant may have a shared static key. When the KDC can be involved in the process of session key establishment, there may be no need for participants to have shared static keys among themselves. In this sense, the availability of a KDC facilitates or simplifies key establishment. A disadvantage of the involvement of a KDC is that compromise of the KDC would result in the compromise of an entire system that employs it. In addition, the KDC may become a bottleneck of the system, due to a heavy load imposed on it.

Notable examples of key establishment relying on a KDC include Kerberos [33] and some versions of KryptoKnight [8, 30].

In some other applications, there may be no KDC available. Instead, there may exist only an "off-line" trusted third party called a certification authority (CA). The main function of a certification authority is to issue certificates on a participant's public key(s) by the use of digital signature. Although a certification authority does not involve itself in the process of key establishment, its existence is implicit in all public key based session key establishment protocols.

11.7 Efficiency

Each application may have its own set of requirements on the efficiency of a key establishment protocol. For example, secure mobile communications generally require a "lightweight" protocol, as a mobile device is usually computationally less powerful than a wired one. As a second example, a network layer security application has far more stringent requirements on the efficiency of key establishment than does an upper layer application.

Factors that contribute to the efficiency of a key establishment protocol include (1) the number of moves (or flows, passes) of messages between two participants, (2) the length of messages communicated between the participants (measured in bits), (3) the computational cost invested by both participants, (4) the size of secure storage, (5) the degree of pre-computation (which is especially important if the protocol is intended to be used with computationally weak devices), and so on. One of the challenges that face a protocol designer is to arrive at a key establishment protocol that would not only minimize the first four factors but also maximize the fifth factor, while maintaining the goals of the protocol.

Optimally efficient protocols that rely on secret key cryptosystems and/or a KDC have been proposed in [8, 24]. Among public key based protocols, Beller and Yacobi's proposals [6] minimize the computational requirements of a less powerful participant. These protocols are particularly suitable for applications where one of the two participants is computationally weak.

11.8 Goals to be Achieved

The main goals of this proposal are to present authenticated key establishment protocols that (1) do not rely on a trust key distribution center or KDC, (2) have a low computational cost, (3) are compact so that the length of each message exchanged is as short as possible, and (4) offer unforgeability and non-repudiation.

In many key transport protocols that rely on secret key cryptosystems, such as those proposed in [4, 8], messages communicated between Alice and Bob are all compact and can be easily fitted into single ATM cells. Some of these protocols do not offer unforgeability or non-repudiation, while the others do so only with the help of a KDC. In other words, these protocols are not suitable for an application where unforgeability and non-repudiation are to be satisfied without relying on a KDC.

Key establishment using public key cryptosystems does not rely on a KDC in achieving unforgeability and non-repudiation. With all currently known public key based key establishment protocols, however, a single payload field of 48 bytes, or of 384 bits, cannot be used to carry unforgeable key materials. To see why this is the case, we take the RSA cryptosystem as an example. In order to maintain a minimal level of security, it is widely believed that the size of an RSA composite should be of at least 512 bits. Thus merely encrypting key materials will result in an expanded outcome that has as many bits as in the RSA composite. (See [31] for a discussion on various data formats for key transport using RSA.) If, in addition, digital signature is involved to achieve unforgeability, the outcome will be even longer. A similar problem occurs with public key cryptographic techniques based on the ElGamal encryption scheme that relies on the discrete logarithm over finite fields.

The ElGamal encryption scheme built on an elliptic curve over a finite field, say $GF(2^{160})$, deserves special attention. With this scheme, a point on the elliptic curve can be compressed so that it occupies only $160+1 = 161$ bits. Thus a single ATM cell may be used to transmit un-authenticated key materials of up to about $384 - 161 = 223$ bits. However, a field of 223 bits is too small to carry a key and a time-varying quantity together with a signature. In other words, elliptic curve based public key cryptography does not provide a solution to the problem of compact and unforgeable key establishment.

12 Description

12.1 Basic Ideas in Using Signcryption for Key Transport

First we present two possible data formats for Alice to transport key materials to Bob, one carrying directly while the other indirectly key materials.

12.1.1 Direct Transport of Key Materials

Figure 2 illustrates a method for directly transferring key materials. It shows a possible combination of parameters: $|p| \geqq 512$, $|q| = 160$, and $|KH.(\cdot)| = 80$. The actual data from Alice to Bob consist of c, r and s, where $c = E_{k_1}(key, TQ)$, $r = KH_{k_2}(key, TQ, other)$ and $s = x/(r + x_a) \bmod q$, where the *key* part contained in (key, TQ) may be used directly as a random session key, TQ may contain a time-varying quantity such as a nonce or a time-stamp or both, and *other* may be composed of the participants' identifiers, public key certificates and other supplementary information. It is preferable for E to act as a length-preserving encryption function so that (key, TQ) and $c = E_{k_1}(key, TQ)$ are of the same length.

Note that if *key* has 64 bits in length, and that TQ requires 32 bits, then $c = E_{k_1}(key, TQ)$ is of 96 bits, and (c, r, s) can be fitted even in a short packet that has only $96+80+160 = 336$ effective bits for data transport. Furthermore, if the quantity TQ is already known to Bob the recipient, then it may be dropped from $c = E_{k_1}(key, TQ)$ to save more bit locations for transferring key materials.

Figure 2: Direct Transport of Key Materials

12.1.2 Indirect Transport of Key Materials

In certain applications, part of a small packet may be used for other purposes, which would leave no room to directly accommodate both a random session key and a time-varying quantity. With such a short packet, we can transport (part of) key materials indirectly. In particular, we may define (c, r, s) as $c = E_{k_1}(TQ)$, $r = KH_{k_2}(TQ, other)$, and $s = x/(r + x_a) \bmod q$. (see Figure 3). The actual session key may be derived from (k_1, k_2) and other materials, through, for instance, the application of a keyed hash function.

Now assume that TQ has 32 bits. Then we can accommodate (c, r, s) using only $32 + 80 + 160 = 272$ bits. In the case where TQ is already known to Bob, the creation and transmission of the c part can be skipped.

Finally we note that in both Figures 2 and 3, a long TQ, say of 64 bits, may need not be encrypted. However, encryption is mandatory for a short TQ, say of $\leqq 40$ bits, in order to reduce the risk of replay attacks.

Figure 3: Indirect Transport of Key Materials

12.2 Signcryption Based Key Establishment

Now we describe in full details how to establish fresh random session keys between two participants Alice and Bob, in such a way that all messages exchanged between the two participants are short and computational costs involved are minimized.

12.2.1 Assumptions

In the following discussions, we assume that system parameters that are common to all participants, and the public and private keys of both Alice and Bob have all been properly set up according to Table 2. In addition, there is a trusted certification authority (CA) that has already issued a public key certificate to each participant. A participant's public key certificate may comply with X.500 certificate format that contains such information as certificate serial number, validity period, the ID of the participant, the public key of the participant, the ID of the CA, the public key of the CA, etc. It would be pointed out that the digital signature scheme used by the CA in creating public key certificates does not have to be one based on ElGamal signature scheme.

Furthermore, we assume that prior to an execution of a key establishment protocol, both participants have already obtained the other participant's public key and its associated certificate issued by the CA, and have checked and are satisfied with the validity of the certificates. The participants may have done so either because they both keep a list of frequently used certificates, or they have obtained and verified the certificates for previous communication sessions.

In describing a key establishment protocol, $key \in_R \{0,1\}^{\ell_k}$ indicates that key is an ℓ_k-bit number chosen uniformly at random. Similarly $NC_b \in_R \{0,1\}^{\ell_n}$ is a nonce chosen by Bob. And TS is a current time-stamp. Typically $\ell_k \geqq 64$, $\ell_n \geqq 40$, and the number of bits in TS may be decided by the accuracy of clock synchronization, as well as by the life span of a message containing the time-stamp. Finally a 64-bit authentication *tag* would be long enough for the purpose of key confirmation in most practical applications.

We consider key establishment both through key material transport and exchange.

12.2.2 Key Transport Protocols

A key transport protocol may use either a nonce or a time-stamp in guaranteeing freshness. The protocol may also transport key materials either directly or indirectly. So there are in total four possible combinations. Table 6 describes two direct key transport protocols, while Table 6 the corresponding two indirect key transport protocols.

The *etc* part may contain data known to both Alice and Bob. Such data may include the participants' names, public keys, public key certificates, protocol serial number, and so on. It may also contain system control information. Note that one of the purposes of sending *tag* is for key confirmation, namely for a participant (Bob) to show the other (Alice) that he does know the new session key. For a less critical application, a time-stamp TS may be transmitted to Bob in clear to further improve the computational efficiency of the protocols. In addition, if both time-stamps and nonces are available in an application, TS may be substituted with a combination of a time-stamp and a nonce.

As can be seen in the tables, protocols that rely on a nonce require one more message move than protocols that rely on a time-stamp.

12.2.3 Key Exchange and Mutual Identification

In the key transport protocols described above, messages from Bob are not involved the creation of a session key. If one wishes that the session key is generated jointly by Alice and

Direct Key Transport Using a Nonce (Protocol DKTUN)		
Alice		**Bob**
	$\Leftarrow \quad NC_b \quad \Leftarrow$	$NC_b \in_R \{0,1\}^{\ell_n}$
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ $c = E_{k_1}(key)$ $r = KH_{k_2}(key, NC_b, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $key = D_{k_1}(c)$ Accept key only if $KH_{k_2}(key, NC_b, etc) = r$
verify tag	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$tag = MAC_{key}(NC_b)$

Direct Key Transport Using a Time-Stamp (Protocol DKTUTS)		
Alice		**Bob**
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ Get a current time-stamp TS $c = E_{k_1}(key, TS)$ $r = KH_{k_2}(key, TS, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $(key, TS) = D_{k_1}(c)$ Accept key only if TS is fresh and $KH_{k_2}(key, TS) = r$
verify tag	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$tag = MAC_{key}(TS)$

Table 6: Direct Key Material Transport with Signcryption

Indirect Key Transport Using a Nonce (Protocol IKTUN)

Alice		**Bob**
	$\Leftarrow \quad NC_b \quad \Leftarrow$	$NC_b \in_R \{0,1\}^{\ell_n}$
$x \in_R [1, \dots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ $key = k_1$ $r = KH_{k_2}(key, NC_b, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $key = k_1$ Accept key only if $KH_{k_2}(key, NC_b, etc) = r$
verify tag	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$tag = MAC_{key}(TS)$

Indirect Key Transport Using a Time-Stamp (Protocol IKTUTS)

Alice		**Bob**
$x \in_R [1, \dots, q-1]$ $(k_1, k_2) = hash(y_b^x \bmod p)$ Get a current time-stamp TS $c = E_{k_1}(TS)$ $r = KH_{k_2}(TS, etc)$ $s = x/(r + x_a) \bmod q$	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$(k_1, k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $TS = D_{k_1}(c)$ Accept (k_1, k_2) only if TS is fresh and $KH_{k_2}(TS, etc) = r$
$key = KH_{k_1,k_2}(TS)$ verify tag	$\Leftarrow \quad tag \quad \Leftarrow$ (optional)	$key = KH_{k_1,k_2}(TS)$ $tag = MAC_{key}(TS, 1)$

Table 7: Indirect Key Material Transport with Signcryption

Bob, there are a few different ways that can be used to accomplish this. Here are some examples: (1) $key^* = KH_{key}(NC_b)$, (2) $key^* = KH_{key}(ID_b)$, and (3) $key^* = KH_{key}(NC_b, ID_b)$, where NC_b is a nonce generated by Bob, ID_b is Bob's identifier, and key^* denotes a session key that is jointly determined by information from both Alice and Bob.

Two common properties shared by the four protocols are: (1) Alice identifies herself to Bob (her message to Bob is fresh and unforgeable even by Bob), (2) Bob authenticates himself to Alice if the last response message *tag* is sent (*tag* is fresh and unforgeable by any third party). The protocols can be modified to achieve *mutual identification*: Alice sends to Bob fresh and unforgeable key materials and vice versa. We take as examples the two protocols for direct key transport. Modifications to the protocols are shown in Table 8. The modified protocols are direct key (material) *exchange* protocols that achieve mutual identification. A resultant $key \oplus key^*$ can be used as a fresh session key jointly generated by both participants.

The other two protocols for for indirect key transport can be modified in a similar way.

12.2.4 Two-Way Communications

For two-way communications, Alice and Bob may need to agree upon a pair of random session keys key_1 and key_2. A simple technique is to employ a pseudo-random number generator or a good hashing function to "extend" key into (key_1, key_2).

13 Advantages

13.1 Non-repudiation, Unforgeability and Identification

Without involving a trusted third party, a key agreement protocol that does not employ digital signature cannot provide identification or non-repudiation, as every message created by one participant can also be created by the other. Nor can the protocol be used in identification.

All the key agreement protocols described in this submission have the properties of non-repudiation and unforgeability, and can be used for identification purposes.

13.2 Message Compactness and Computational Efficiency

Protocols built in the signature-then-encryption approach can achieve non-repudiation and unforgeability. Now we compare the efficiency of such protocols with that of ours.

Every message in the key transport protocols proposed in this submission is compact and can be carried by a short packet such as a single ATM cell. In terms of computational cost, it takes one modular exponentiation on Alice's side, and two modular exponentiations on Bob's side which can be reduced to 1.17 exponentiations (on average) when Shamir's method for fast evaluation of the product of several exponentials with the same modulo (see Appendix B). As for pre-computation, the exponentiation by Alice, $y_b^x \bmod p$, can be done prior to the start of an execution of a protocol, only if Alice knows beforehand that she is going to communicate with Bob at a later time. In what follows we compare our protocols with two different sets of session key establishment protocols which serve as representatives employing the signature-then-encryption approach.

Direct Key Exchange Using a Nonce (Protocol DKEUN)

Alice		**Bob**
	$\Leftarrow \quad NC_b \quad \Leftarrow$	$NC_b \in_R \{0,1\}^{\ell_n}$
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1,\ldots,q-1]$ $(k_1,k_2) = hash(y_b^x \bmod p)$ $c = E_{k_1}(key)$ $r = KH_{k_2}(key, NC_b, etc)$ $s = x/(r+x_a) \bmod q$	$\Rightarrow \quad c,r,s \quad \Rightarrow$	(k_1,k_2) $= hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $key = D_{k_1}(c)$ Accept key only if $KH_{k_2}(key, NC_b, etc) = r$
(k_1^*, k_2^*) $= hash((y_b \cdot g^{r^*})^{s^* \cdot x_a} \bmod p)$ $key^* = D_{k_1^*}(c^*)$ Accept key^* only if $KH_{k_2^*}(key^*, key, etc) = r^*$	$\Leftarrow \quad c^*, r^*, s^* \quad \Leftarrow$	$key^* \in_R \{0,1\}^{\ell_k}$ $x^* \in_R [1,\ldots,q-1]$ $(k_1^*, k_2^*) = hash(y_a^{x^*} \bmod p)$ $c^* = E_{k_1^*}(key^*)$ $r^* = KH_{k_2^*}(key^*, key, etc)$ $s^* = x^*/(r^*+x_b) \bmod q$
$tag = MAC_{key \oplus key^*}(NC_b)$	$\Rightarrow \quad tag \quad \Rightarrow$ (optional)	verify whether $tag = MAC_{key \oplus key^*}(NC_b)$

Direct Key Exchange Using a Time-Stamp (Protocol DKEUTS)

Alice		**Bob**
$key \in_R \{0,1\}^{\ell_k}$ $x \in_R [1,\ldots,q-1]$ $(k_1,k_2) = hash(y_b^x \bmod p)$ Get a current time-stamp TS $c = E_{k_1}(key, TS)$ $r = KH_{k_2}(key, TS, etc)$ $s = x/(r+x_a) \bmod q$	$\Rightarrow \quad c,r,s \quad \Rightarrow$	$(k_1,k_2) = hash((y_a \cdot g^r)^{s \cdot x_b} \bmod p)$ $(key, TS) = D_{k_1}(c)$ Accept key only if TS is fresh and $KH_{k_2}(key, TS) = r$
(k_1^*, k_2^*) $= hash((y_b \cdot g^{r^*})^{s^* \cdot x_a} \bmod p)$ $(key^*, TS^*) = D_{k_1^*}(c^*)$ Accept key^* only if TS^* is fresh and $KH_{k_2^*}(key^*, TS^*, key, etc) = r^*$	$\Leftarrow \quad c^*, r^*, s^* \quad \Leftarrow$	$key^* \in_R \{0,1\}^{\ell_k}$ $x^* \in_R [1,\ldots,q-1]$ $(k_1^*, k_2^*) = hash(y_a^{x^*} \bmod p)$ Get a current time-stamp TS^* $c^* = E_{k_1^*}(key^*, TS^*)$ $r^* = KH_{k_2^*}(key^*, TS^*, key, etc)$ $s^* = x^*/(r^*+x_b) \bmod q$
$tag = MAC_{key \oplus key^*}(TS)$	$\Rightarrow \quad tag \quad \Rightarrow$ (optional)	verify whether $tag = MAC_{key \oplus key^*}(TS)$

Table 8: Direct Key Material Exchange Achieving Mutual Identification

13.2.1 Comparison with ATM Forum Proposals

First we consider a proposed standard related to security in ATM. The current version of Phase I ATM Security Specification [14, 45] contains two key material exchange protocols. One involves three and the other two moves or flows of messages (see Sections 6.1.1 and 6.1.2 of [14]). These two protocols have been largely based on X.509 [29]. To describe the protocols defined in the Specification [14], we use the following symbols and abbreviations which are essentially the same as those defined the document.

1. ID_a is the (distinguished) name of Alice.
2. T_a is a time-stamp generated by Alice, consisting of a 4-byte coordinated universal time and a 4-byte sequential number.
3. R_a a nonce generated by Alice.
4. $Enc_{K_b}(\cdot)$ denotes encryption by Alice using either a secret key or a public key algorithm.
5. $ConfPar_a$ contains key materials from Alice.
6. $Sig_{K_a}(\cdot)$ denotes signature generation by Alice.
7. $cert_a$ denotes Alice's public key certificate and is used in the three-way protocol.
8. $SecNeg_a$ carries information on types of security services to be provided, algorithm and protocol options available and parameters requested for a connection. It is used only in the three-way protocol.
9. *SecOpt* is generated by Alice, carrying information similar to that contained in $SecOpt$, although it is used only in the two-way protocol.
10. ID_b, T_b, R_b, $Enc_{K_a}(\cdot)$, $ConfPar_b$, $Sig_{K_b}(\cdot)$, $SecNeg_b$ and $cert_b$ are all associated with Bob and defined similarly.
11. $\{X\}$ indicates that X is optional.

With the aid of the above symbols, we summarize in Table 9 the two protocols proposed in Phase I ATM Security Specification.

It is stated in Phase I ATM Security Specification that the two key material exchange protocols can be implemented either in secret key (symmetric) cryptography or public key (asymmetric) cryptography. What we are interested in the present work is the latter, namely, the case when the two protocols are implemented in public key cryptography. Leaving out some of the technical details which are not directly relevant to our analysis, it becomes clear that both protocols follow the traditional signature-then-encryption approach. Furthermore, we can see that the three-way key material exchange protocol is based on nonces, while the two-way protocol is based on a time-stamp (together with a nonce). Thus the three-way protocol achieves similar goals to those by our protocol DKEPUN described in Table 8, and the two-way protocol achieves similar goals to those by our protocol DKEPUTS described in the same table. As is expected, our signcryption-based protocols DKEPUN and DKEPUTS are significantly more efficient than their respective counterparts proposed by the ATM Forum, both in terms of computational cost and message overhead. A detailed comparison can be easily worked out by the use of Tables 4 and 5.

	Three-Way Key Material Exchange Protocol	
Alice (Initiator)		**Bob (Respondent)**
$\Rightarrow$	$ID_a, \{ID_b\}, R_a, SecNeg_a, \{Cert_a\}$	$\Rightarrow$
$\Leftarrow$	$ID_a, ID_b, SecNeg_b, \{Cert_b\},$ $\{R_a, R_b, \{Enc_{K_a}(ConfPar_b)\},$ $Sig_{K_b}(hash(ID_a, ID_b, R_a, R_b, SecNeg_a, SecNeg_b, \{ConfPar_b\}))\}$	$\Leftarrow$
$\Rightarrow$	$\{ID_a, ID_b, R_b, \{Enc_{K_b}(ConfPar_a)\},$ $Sig_{K_a}(hash(ID_a, ID_b, R_b, \{ConfPar_a\}))\}$	$\Rightarrow$
	Two-Way Key Material Exchange Protocol	
Alice (Initiator)		**Bob (Respondent)**
$\Rightarrow$	$ID_a, ID_b, SecOpt, \{T_a, R_a, \{Enc_{K_b}(ConfPar_a)\},$ $Sig_{K_a}(Hash(ID_a, ID_b, T_a, R_a, SecOpt, \{ConfPar_a\}))\}$	$\Rightarrow$
$\Leftarrow$	$\{ID_a, ID_b, R_a, \{Enc_{K_a}(ConfPar_b)\},$ $Sig_{K_b}(Hash(ID_a, ID_b, R_a, \{ConfPar_b\}))\}$	$\Leftarrow$

Table 9: Key Material Exchange Protocols Proposed by ATM Forum

13.2.2 Comparison with Beller-Yacobi Protocol

The next protocol we examine is an efficient proposal by Beller and Yacobi [6]. Their protocol is briefly summarized in Table 10, using notations consistent with those for signcryption schemes. As is the case for our proposals based on signcryption, here it is assumed too that public key certificates have already been transferred prior to an execution of the protocol. In Beller-Yacobi protocol, Alice uses ElGamal signature scheme to sign a message, and cubic RSA to encrypt the message before delivering it to Bob. Bob holds the matching cubic RSA decryption key and hence can extract the message. The number of modular exponentiations done by Alice is one (for signature generation), and by Bob is four (one for decrypting cubic RSA and three for verifying Alice's digital signature). Shamir's technique for fast evaluation of the product of several exponentials with the same modulo can also be used to speed-up the verification of ElGamal signature by Bob. More specifically, the cost for computing the product of modulo three exponentiations on Bob's side can be reduced to 1.25 modulo exponentiations on average. It is important to note that since the decryption operation for the cubic RSA on Bob's side involves an exponentiation with a full size exponent, it can be very time-consuming, especially when the RSA composite is large. Table 11 indicates that our protocols are indeed advantageous compared to Beller-Yacobi protocol.

14 Security Assessment

As our key establishment protocols described in Tables 6 and 7 are essentially message transport schemes using signcryption, security of key materials are guaranteed by the security of the signcryption scheme against chosen message attacks [54, 55]. After the successful establishment of a session key, Alice convinces Bob of her identify (the message from Alice is fresh and unforgeable even by Bob). In contrast, Bob can authenticate himself to Alice by sending a response message *tag* which is fresh and unforgeable by a third party (but can be generated by Alice). The four protocols can be modified using a method shown in Table 8 in order to achieve mutual identification, at the expense of more computation and

Alice		**Bob**
$K \in_R \{0,1\}^{\ell_k}$ $c_1 = K^3 \bmod n_B$	$\Rightarrow c_1 \Rightarrow$	Extract K from c_1 by using the decryption key associated with the RSA composite n_B
Decrypt c_2 and verify the format of the message	$\Leftarrow c_2 \Leftarrow$	Choose a random m $c_2 = E_K(m, 0^t)$
Compute ElGamal signature (v, w) on (m, etc) $c_3 = E_K(v, w, etc)$	$\Rightarrow c_3 \Rightarrow$	Decrypt c_3 and verify (v, w)

Table 10: Beller-Yacobi Authenticated Key Transport Protocol

Protocols	Comp. Cost # of exp.	Pre-Comp. by Alice	Longest Message (typical example)
Beller-Yacobi	$1 + 2.25^*$	Yes	$\geqq \lvert n_B \rvert$ bits ($\geqq$ 512 bits)
DKTUN & DKTUTS	$1 + 1.17$	Yes$^+$	$\leqq$ 384 bits ($<$ 384 bits)
IKTUN	$1 + 1.17$	Yes$^+$	$<$ 384 bits (240 bits)
IKTUTS	$1 + 1.17$	Yes$^+$	$<$ 384 bits (280 bits)

* Including an RSA decryption with a full size exponent.
$^+$ Only when Alice knows whom to communicate with.

Table 11: Comparison with Beller-Yacobi Protocol

message exchanges.

Freshness of a session key is assured through the use of a nonce or a time-stamp. When *tag* is sent, both Alice and Bob are assured that the other participant does know the fresh random session key. The protocols do not rely on a KDC. In addition, key materials transported from Alice to Bob are unforgeable, even by Bob the recipient. The materials are also non-repudiateable by Alice. In an event when Alice denies the fact that she was the person who created certain key materials, Bob can ask for help from a third party called a judge. Bob and the judge may follow a zero-knowledge protocol in settling the dispute [54, 55]. Similar discussions on non-repudiation are applicable to Bob for a modified protocol with mutual identification.

15 Known Limitations

Limitations with the key agreement protocols are similar to those with the underlying signcryption schemes (see Section 9). Some additional discussions on "forward secrecy" follow.

15.1 Forward Secrecy

A key establishment protocol is said to offer forward secrecy with respect to a participant if compromise of the participant's long term secret key does not result in compromise of past session keys. Clearly a key establishment protocol based on a shared static key between two participants cannot offer forward secrecy.

Among protocols that are based on public key cryptography and offer forward secrecy with respect to both participants are those derived from the Diffie-Hellman key establishment protocol (see for example protocols proposed in [20, 25, 9]). Adding to these is Beller-Yacobi protocol [6] which offers forward secrecy with respect to Alice the sender (but not with respect to Bob the receiver). In contrast, the signcryption based key transport protocols proposed in this submission do not offer forward secrecy with respect to either participant.

However, it is our view that one cannot categorically claim that a key establishment protocol with forward secrecy is better than one without. Rather one should take into account the additional computational and communication overhead involved in providing forward secrecy.

There are basically two approaches that may be employed in containing potential damages due to compromise of a long term secret key. The first is to design a key establishment protocol that offers forward secrecy and hence can tolerate compromise of the key. The second is to find a way to make the key less compromiseable. As will be shown immediately, the second approach seems far more economical than the first one in terms of extra computational cost involved.

Before proceeding to a discussion on how to protect a participant's long term secret key from being compromised, we note that there are mainly two possible threats to the long term secret key: accidental loss and, more serious, theft. It turns out that both threats can be effectively thwarted via such means as secret sharing, either in a mathematical [50] or physical sense.

To illustrate how simple and effective a secret sharing method is against the theft and accidental loss of a long term secret key, we take a look at Alice's long term secret key x_a. What Alice can do is to choose a random number $x_{a,1}$, calculate $x_{a,2} = x_a \oplus x_{a,1}$, and then store $x_{a,1}$ and $x_{a,2}$ in two different secure locations. These secure locations can be logically separate secure compartments in Alice's computer system, two physical devices (say, one is a tamper-resistant smart card, the other a PC with a lock), or a combination of logically and physically secure facilities. One can see that Alice can readily recover x_a from $x_{a,1}$ and $x_{a,2}$ by computing $x_a = x_{a,1} \oplus x_{a,2}$, and the extra computational cost to be invested by Alice is negligible. However, for an attacker or intruder to successfully steal x_a, he has to break into both secure locations, a task that would be twice as hard as breaking into one of them.

The above method is called a 2 out of 2, or $(2, 2)$ threshold secret sharing scheme. It can be extended to $(3, 3)$, $(4, 4)$ and so on. More generally, Alice can use a t out of n, or (t, n) threshold secret sharing scheme, where $n \geqq t$, in safeguarding her long term secret key x_a. An example of (t, n) threshold secret sharing schemes is Shamir's scheme based on polynomial interpolation on a finite field [50]. The computational cost involved in a (t, n) secret sharing scheme is marginal when compared with an exponentiation modulo a large integer. No information, in an information-theoretic sense, on a long term secret key dispersed in a (t, n) threshold scheme is leaked to an attacker even if he has managed to break into up to $t - 1$ of the secure locations. An added benefit is that Alice can still reconstruct x_a even when up to $n - t$ secure locations are un-recoverably damaged.

16 Patent Issues

Monash University has lodged applications for patents both in Australia and USA. "Reasonable and non-discriminatory" patent licensing is available from Monash.

References

[1] K. Araki, K. Satoh, and S. Miura. Overview of elliptic curve cryptography. In *Proceedings of 1998 International Workshop on Practice and Theory in Public Key Cryptography (PKC'98)*, volume 1431 of *Lecture Notes in Computer Science*, pages 29–49, Berlin, New York, Tokyo, 1998. Springer-Verlag.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Berlin, New York, Tokyo, 1996. Springer-Verlag.

[3] M. Bellare, M. Jakobsson, and M. Yung. Round-optimal zero-knowledge arguments based on any one-way function. In *Advances in Cryptology - EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 280–305, Berlin, New York, Tokyo, 1997. Springer-Verlag.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Berlin, New York, Tokyo, 1993. Springer-Verlag.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 62–73, New York, November 1993. The Association for Computing Machinery.

[6] M. Beller and Y. Yacobi. Fully-fledged two-way public key authentication and key agreement for low cost terminals. *Electronic Letters*, 30:999–1001, 1993.

[7] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of efficient provably secure two-way authentication protocols. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 44–61, Berlin, New York, Tokyo, 1992. Springer-Verlag.

[8] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. The KryptoKnight family of authentication and key distribution protocols. *IEEE/ACM Transactions on Networking*, 1995.

[9] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocol and their security analysis, 1997. (submission to IEEE P1363).

[10] E. Brickell and K. McCurley. Interactive identification and digital signatures. *AT&T Technical Journal*, pages 73–86, November/Decmber 1991.

[11] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[12] D. Chaum. Zero-knowledge undeniable signatures. In *Advances in Cryptology - EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 458–464, Berlin, New York, Tokyo, 1990. Springer-Verlag.

[13] M. Chen and E. Hughes. Protocol failures related to order of encryption and signature: Computation of discrete logarithms in RSA groups, April 1997. (Draft).

[14] The ATM Forum Technical Committee. Phase I ATM security specification (draft), July 1997. (ATM Forum BTD-SECURITY-01.03).

[15] The ATM Forum Technical Committee. Security framework for ATM networks (draft), April 1997. (ATM Forum BTD-FRWK-01.00).

[16] D. Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology - EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 153–165, Berlin, New York, Tokyo, 1996. Springer-Verlag.

[17] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. In *Advances in Cryptology - EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 1–9, Berlin, New York, Tokyo, 1996. Springer-Verlag.

[18] Y. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, 1994.

[19] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):472–492, 1976.

[20] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and authenticated key exchange. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[21] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[22] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[23] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptively chosen message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[24] L. Gong. Efficient network authentication protocols: Lower bounds and optimal implementations. *Distributed Computing*, 9(3):131–145, 1995.

[25] D. Harkins and D. Carrel. The resolution of ISAKMP with Oakley, March 1998. (Internet-draft: draft-ietf-ipsec-isakmp-oakley-07.txt).

[26] S. Hirose and S. Yoshida. An authenticated Diffie-Hellman key agreement protocol secure against active attacks. In H. Imai and Y. Zheng, editors, *Public Key Cryptography'98*, volume 1431 of *Lecture Notes in Computer Science*, pages 135–149, Berlin, New York, Tokyo, 1998. Springer-Verlag.

[27] P. Horster, M. Michels, and H. Petersen. Meta-ElGamal signature schemes. In *Proceedings of the second ACM Conference on Computer and Communications Security*, pages 96–107, New York, November 1994. The Association for Computing Machinery.

[28] IEEE. *Standard Specifications For Public Key Cryptography (P1363)*. The Institute of Electrical and Electronics Engineers, August 1998. (Draft Version 6, `http://grouper.ieee.org/groups/1363/`).

[29] ITU. Information technology - open systems interconnection - the directory: Authentication framework. Recommendation X.509, International Telecommunications Union, 1993.

[30] P. Janson, G. Tsudik, and M. Yung. Scalability and flexibility in authentication services: The KryptoKnight approach. In *Proceedings of INFOCOM'97*. IEEE, 1997.

[31] D. Johnson and S. Matyas. Asymmetric encryption: Evolution and enhancements. *CryptoBytes*, 2(1):1–6, 1996. (available at `http://www.rsa.com/`).

[32] J. Kilian and E. Petrank. An efficient non-interactive zero-knowledge proof system for NP with general assumption. *Electronic Colloquium on Computational Complexity*, Reports Series(TR95-038), 1995. (available at `http://www.eccc.uni-trier.de/eccc/`).

[33] J. Kohl and B. C. Neuman. The Kerberos network authentication services (v5). Request for Comments RFC 1510, IETF, 1993.

[34] A. Lenstra. Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. In *Information Security and Privacy – Proceedings of ACISP'97*, volume 1270 of *Lecture Notes in Computer Science*, pages 127–138, Berlin, New York, Tokyo, 1997. Springer-Verlag.

[35] A. K. Lenstra and H. W. Lenstra. *Algorithms in Number Theory*, volume A of *Handbook in Theoretical Computer Science*, chapter 12, pages 673–715. Elsevier and the MIT Press, 1990.

[36] T. Matsumoto and H. Imai. On the key predistribution systems: A practical solution to the key distribution problem. In *Advances in Cryptology - CRYPTO'87*, volume 239 of *Lecture Notes in Computer Science*, pages 185–193, Berlin, New York, Tokyo, 1987. Springer-Verlag.

[37] A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, IT-39(5):1639–1646, 1993.

[38] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[39] National Bureau of Standards. Data encryption standard. Federal Information Processing Standards Publication FIPS PUB 46, U.S. Department of Commerce, January 1977.

[40] National Institute of Standards and Technology. Digital signature standard (DSS). Federal Information Processing Standards Publication FIPS PUB 186, U.S. Department of Commerce, May 1994.

[41] National Institute of Standards and Technology. Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1, U.S. Department of Commerce, April 1995.

[42] K. Nyberg and R. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes and Cryptography*, 7(1/2):61–81, 1996.

[43] A. Odlyzko. The future of integer factorization. *CryptoBytes*, 1(2):5–12, 1995. (available at `http://www.rsa.com/`).

[44] H. K. Orman. The Oakley key determination protocol, July 1997. (Internet-draft: draft-ietf-ipsec-oakley-02.txt).

[45] M. Peyravian and T. Tarman. Asynchronous transfer mode security. *IEEE Network*, 11(3):34–40, May/June 1997.

[46] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology - EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Berlin, New York, Tokyo, 1996. Springer-Verlag.

[47] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 1998. (to appear).

[48] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, October 1997.

[49] C. P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–251, Berlin, New York, Tokyo, 1990. Springer-Verlag.

[50] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[51] A. Shamir. RSA for paranoids. *CryptoBytes*, 1(3):1–4, 1995. (available at `http://www.rsa.com/`).

[52] N. Smart. A message posted to the number theory list `<nmbrthry@listserv.nodak.edu>`, September 1997.

[53] Y. Zheng. Improved public key cryptosystems secure against chosen ciphertext attacks. Technical Report 94-1, University of Wollongong, Australia, January 1994.

[54] Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption). In *Advances in Cryptology - CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179, Berlin, New York, Tokyo, 1997. Springer-Verlag.

[55] Y. Zheng. Signcryption and its applications in efficient public key solutions. In *Information Security — Proceedings of 1997 Information Security Workshop (ISW'97)*, volume 1396 of *Lecture Notes in Computer Science*, pages 291–312, Berlin, New York, Tokyo, 1998. Springer-Verlag. (an invited talk).

[56] Y. Zheng and H. Imai. Compact and unforgeable session key establishment over an ATM network. In *Proceedings of IEEE INFOCOM'98*, pages 411–418, San Francisco, 1998. IEEE.

[57] Y. Zheng and H. Imai. Efficient signcryption schemes on elliptic curves. In *Global IT Security — Proceedings of the IFIP TC11 14th International Conference on Information Security (IFIP/SEC'98)*, pages 75–84, Vienna and Budapest, August 1998. International Federation for Information Processing (IFIP).

[58] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL - a one-way hashing algorithm with varialbe length of output. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology - AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104, Berlin, New York, Tokyo, 1993. Springer-Verlag.

[59] Y. Zheng and J. Seberry. Immunizing public key cryptosystems against chosen ciphertext attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):715–724, June 1993.

A Extensions to Elliptic Curves

The shortened digital signature, signcryption and key agreement schemes can all be built on secure elliptic curves [54, 55, 57, 56]. What follows is an outline of elliptic curve based shortened digital signature and corresponding signcryption schemes. Description of the corresponding elliptic curve based key agreement schemes is straightforward, and hence omitted.

A.1 Description

A.1.1 Elliptic Curve Cryptography

The ordinal ElGamal public key encryption and digital signature schemes are defined on finite fields. In 1985 Neal Koblitz from the University of Washington and Victor Miller then with IBM observed that discrete logarithm on elliptic curves over finite fields appeared to be intractable and hence ElGamal's encryption and signature schemes have natural counterparts on these curves. (See documents on IEEE P1363 [28] for more detailed information on this topic.)

Let $GF(p^m)$ be the finite field of p^m elements, where p is a prime and m an integer, an elliptic curve over $GF(p^m)$ is defined as the set of solutions (x, y), where $x, y \in GF(p^m)$, to a cubic equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

with $a_1, a_2, a_3, a_4, a_6 \in GF(p^m)$, together with a special point $\mathcal{O}$ called the *point at infinity*. In cryptographic practice, we are particularly interested in (1) elliptic curves over $GF(2^m)$ with $m > 150$, and (2) elliptic curves over $GF(p)$ with p a large prime. Hence these two types of elliptic curves deserve a closer look.

For $GF(2^m)$, the cubic equation for an elliptic curve takes the form of

$$\begin{cases} y^2 + cy = x^3 + ax + b, & \text{with } a, b, c \in GF(2^m),\ c \neq 0 \text{ and } j\text{-variant } 0 \\ \text{or} & \\ y^2 + xy = x^3 + ax^2 + b, & \text{with } a, b \in GF(2^m),\ b \neq 0 \text{ and } j\text{-variant not } 0 \end{cases} \tag{1}$$

And for $GF(p)$, $p > 3$, the cubic equation takes the form of

$$y^2 = x^3 + ax + b, \qquad \text{with } a, b \in GF(p) \text{ and } 4a^3 + 27b^2 \neq 0 \tag{2}$$

An elliptic curve over $GF(p^m)$ forms an abelian group under an addition on the points given by the "tangent and chord method". To be precise, this group should be called an *elliptic curve group* over $GF(p^m)$. In this proposal we follow a common practice to call the group an elliptic curve over $GF(p^m)$.

The addition on an elliptic curve only involves a few arithmetic operations in $GF(p^m)$, and hence is efficient. Taking an elliptic curve C on $GF(p)$ with $p > 3$ as an example, the addition follows the rules specified below:

1. $\mathcal{O} + \mathcal{O} = \mathcal{O}$.
2. $P + \mathcal{O} = P$ for all $P = (x, y) \in C$. Namely, C has $\mathcal{O}$ as its identity element.
3. $P + Q = \mathcal{O}$ for all $P = (x, y) \in C$ and $Q = (x, -y)$. Namely, the inverse of (x, y) is simply $(x, -y)$.

4. Adding two distinct points — for all $P = (x_1, y_1) \in C$ and $Q = (x_2, y_2) \in C$ with $x_1 \neq x_2$, $P + Q = (x_3, y_3)$ is defined by

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

where $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$.

5. Doubling a point — for any $P = (x, y) \in C$ with $y \neq 0$, $2P = (x^*, y^*)$ is defined by

$$\begin{aligned} x^* &= \lambda^2 - 2x \\ y^* &= \lambda(x - x^*) - y \end{aligned}$$

where $\lambda = \frac{3x^2 + a}{2y}$.

Adding and doubling points on an elliptic curve C over $GF(2^m)$ are defined in a similar way.

Excluding the point at infinity $\mathcal{O}$, every point $P = (x, y)$ on an elliptic curve C over $GF(p^m)$ can be represented as (or "compressed" to) $P = (x, \tilde{y})$ where $\tilde{y}$ is a single bit:

1. if $x = 0$, then $\tilde{y} = 0$.

2. if $x \neq 0$, then $\tilde{y}$ is the parity of y when it is viewed as an integer.

An advantage of compressed representation of a point is that when a compressed point is stored internally in a computer or communicated over a network, it takes only one bit more than half of the bits required for storing or transmitting its uncompressed counterpart. This advantage, however, is not for free: recovering the y-coordinate from a compressed point involves a few arithmetic operations in the underlying finite field.

A result due to Hasse states that the order $\#C$ of an elliptic curve C over $GF(p^m)$, i.e., the number of elements in the group, satisfies the following condition

$$\#C = p^m + 1 - t, \qquad \text{with } |t| \leqq 2\sqrt{p^m} \tag{3}$$

where t is called the *trace of the elliptic curve* C, or to be more precise, the *trace of the Frobenius endomorphism* of C. Structurally, C is known to be isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, where both n_1 and n_2 are integers, $n_2 | n_1$, $n_2 | (p^m - 1)$ and $\mathbb{Z}_n$ denotes the modular ring of n elements.

Let G be a point on an elliptic curve C over $GF(p^m)$. The order of G is the smallest integer q such that $qG = \mathcal{O}$. For an integer e, the e multiple of G, namely eG, can be readily computed by using a method similar to the "square-and-multiply" for exponentiation in $GF(p)$. The inverse problem corresponding to the computation of a multiple of a point is that given two points G and P in C, one is asked to find an integer e such that $P = eG$, provided that such an integer exists. This is known as the elliptic curve discrete logarithm problem. When the order q of G contains a large prime factor, say of size at least 2^{160}, it is believed that the elliptic curve discrete logarithm problem is infeasible to solve. All elliptic curve based cryptosystems hinge their security on the (purported) hardness of the elliptic curve discrete logarithm problem.

In light of recent developments in cracking the elliptic curve discrete logarithm problem [37, 52, 48, 1], however, one should be very cautious in designing a cryptosystem based on the elliptic curve discrete logarithm problem. In particular, it has been shown in [37] that the discrete logarithm problem on a supersingular elliptic curve is not more difficult to solve than the discrete logarithm problem in a finite field. Supersingular elliptic curves on $GF(p^m)$ are curves whose trace t satisfies the condition of

$$t = \pm\sqrt{i \cdot p^m} \quad \text{with } i = 0, 1, 2, 3, \text{or } 4.$$

A more recent breakthrough is dramatic indeed: Nigel Smart at HP Labs in UK, and Takakazu Satoh and Kiyomichi Araki in Japan announced that they have independently broken the discrete logarithm problem on anomalous elliptic curves over $GF(p)$ [52, 48]. An anomalous elliptic curves over $GF(p)$ is a curve whose trace is 1, i.e., a curve that has exactly p points. In their preprint, Satoh and Araki present an algorithm that solves the elliptic curve discrete logarithm problem for trace 1 in $O((\log p)^3)$ steps.

Let us assume, optimistically, that the effectiveness of the algorithms reported in [37, 52, 48] is limited to supersingular and anomalous elliptic curves. Then the fastest known algorithm for the discrete logarithm problem on other elliptic curves appears to take time in the order of $O(\sqrt{p^m})$ which grows exponentially with the size of the elliptic curve group. In other words, on elliptic curves which are not supersingular or with trace 1, the discrete logarithm problem appears to share a similar degree of hardness with the discrete logarithm problem in a sub-group of comparable order modulo a large prime. This point is the origin of signcryption schemes to be introduced in the next section.

A.1.2 Elliptic Curve Signcryption Schemes

As we mentioned earlier, ElGamal public key encryption and digital signature schemes and their variants can all be extended to elliptic curves in a straightforward way. For the sake of completeness, Table 12 summarizes an elliptic curve version of the Digital Signature Standard or DSS [40], together with its shortened variants. The elliptic curve DSS will be called ECDSS, and its two shortened versions SECDSS1 and SECDSS2 respectively. Note that in the computation of $r = (vG) \bmod q$ with ECDSS, $vG = K$ which is a point on an elliptic curve is viewed as an integer. Similarly, in $r = hash(vG, m)$ with SECDSS1 and SECDSS2, vG is viewed as a binary string. Also note that instead of vG, one may involve only its x-coordinate in the computation of r, as the y-coordinate carries essentially only one bit of information and hence may be excluded.

Parameters for elliptic curve based signcryption schemes are summarized in table 13, and two signcryption schemes built on SECDSS1 and SECDSS2 are described in Table 14. These signcryption schemes are called ECSCS1 and ECSCS2 respectively. Similarly to elliptic curve signature schemes described in Table 12, points on an elliptic curve, namely vP_a, $uP_a + urG$ and $uG + urP_a$, are regarded as binary strings when involved in hashing. The *bind_info* part in the computation of r contains, among other data items, identification information of Bob the recipient such as his public key or public key certificate.

A.2 Advantages

A.2.1 Saving in computational cost

With the signature-then-encryption based on SECDSS1 or SECDSS2 and elliptic curve ElGamal encryption, the number of computations of multiples of points is three, both for the process of signature-then-encryption and that of decryption-then-verification.

Shortened schemes	Signature (r, s) on a message m	Verification of signature	Length of signature
ECDSS	$r = (vG) \bmod q$ $s = \frac{hash(m)+v_a r}{v} \bmod q$	$K = s'(hash(m)G + rP_a)$ where $s' = \frac{1}{s} \bmod q$, check whether $K \bmod q = r$	$2\|q\|$
SECDSS1	$r = hash(vG, m)$ $s = \frac{v}{r+v_a} \bmod q$	$K = s(P_a + rG)$ check whether $hash(K, m) = r$	$\|hash(\cdot)\| + \|q\|$
SECDSS2	$r = hash(vG, m)$ $s = \frac{v}{1+v_a \cdot r} \bmod q$	$K = s(G + rP_a)$ check whether $hash(K, m) = r$	$\|hash(\cdot)\| + \|q\|$

C: an elliptic curve over $GF(p^m)$, either with $p \geqq 2^{150}$ and $m = 1$ or $p = 2$ and $m \geqq 150$ (public to all).
q: a large prime whose size is approximately of $|p^m|$ (public to all).
G: a point with order q, chosen randomly from the points on C (public to all).
$hash$: a one-way hash function (public to all).
v: a number chosen uniformly at random from $[1, \ldots, q-1]$.
v_a: Alice's private key, chosen uniformly at random from $[1, \ldots, q-1]$.
P_a: Alice's public key ($P_a = v_a G$, a point on C).

Table 12: Elliptic Curve DSS and Its Shortened and Efficient Variants

Parameters public to all: C — an elliptic curve over $GF(p^m)$, either with $p \geqq 2^{150}$ and $m = 1$ or $p = 2$ and $m \geqq 150$ (public to all). q — a large prime whose size is approximately of $\|p^m\|$ (public to all). G — a point with order q, chosen randomly from the points on C (public to all). $hash$ — a one-way hash function whose output has, say, at least 128 bits. KH — a keyed one-way hash function. (E, D) — the encryption and decryption algorithms of a private key cipher.
Alice's keys: v_a — Alice's private key, chosen uniformly at random from $[1, \ldots, q-1]$. P_a — Alice's public key ($P_a = v_a G$, a point on C).
Bob's keys: v_b — Bob's private key, chosen uniformly at random from $[1, \ldots, q-1]$. P_b — Bob's public key ($P_b = v_b G$, a point on C).

Table 13: Parameters for Elliptic Curve Signcryption

Signcryption of m by Alice the Sender		**Unsigncryption of (c, r, s) by Bob the Recipient**
$v \in_R [1, \ldots, q-1]$ $(k_1, k_2) = hash(vP_b)$ $c = E_{k_1}(m)$ $r = KH_{k_2}(m, bind\_info)$ $s = \frac{v}{r+v_a} \bmod q$ if SECDSS1 is used, or $s = \frac{v}{1+v_a \cdot r} \bmod q$ if SECDSS2 is used.	$\Rightarrow \quad c, r, s \quad \Rightarrow$	$u = sv_b \bmod q$ $(k_1, k_2) = hash(uP_a + urG)$ if SECDSS1 is used, or $(k_1, k_2) = hash(uG + urP_a)$ if SECDSS2 is used. $m = D_{k_1}(c)$ Accept m only if $KH_{k_2}(m, bind\_info) = r$

Table 14: Implementations of Signcryption on Elliptic Curves

We note that the "square-and-multiply" method for fast exponentiation can be adapted to a "doubling-and-addition" method for the fast computation of a multiple of a point on an elliptic curve. Namely a multiple can be obtained in about $1.5|q|$ point additions.

Among the three multiples for decryption-then-verification, two are used in verifying a signature. More specifically, these two multiples are spent in computing $e_1G + e_2P_a$ for two integers e_1 and e_2. Shamir's technique for fast computation of the product of multiple exponentials with the same modulo can be adapted to the fast computation of $e_1G + e_2P_a$. Thus on average, the computational cost for $e_1G + e_2P_a$ is $(1 + 3/4)|q|$ point additions, or equivalently 1.17 point multiples. That is, the number of point multiples involved in decryption-then-verification can be reduced from 3 to 2.17. Consequently, the combined computational cost of the sender and the recipient is 5.17 point multiples..

In contrast, with ECSCS1 and ECSCS2, the number of point multiples is one for the process of signcryption and two for that of unsigncryption respectively. Applying Shamir's technique, one reduces the computational cost of unsigncryption from 2 multiples to 1.17 on average. The total average computational cost for signcryption is therefore 2.17 point multiples. This represents a

$$\frac{5.17 - 2.17}{5.17} = 58\%$$

reduction in average computational cost.

A.2.2 Saving in communication overhead

To simplify our discussions, we assume that $|q| \approx |p^m|$. Namely the order q of G is of comparable size to p^m. In addition we assume that $|hash(\cdot)| = |KH.(\cdot)| = \frac{1}{2}|q|$. Furthermore, we assume that a point on an elliptic curve is represented in a compressed way.

Under these reasonable assumptions, the communication overhead measured in bits is $|hash(\cdot)| + |q| + |p^m + 1| \approx |hash(\cdot)| + 2|q|$ for signature-then-encryption based on SECDSS1 or SECDSS2 and elliptic curve ElGamal encryption, and $|KH.(\cdot)| + |q|$ for the two signcryption schemes ECSCS1 and ECSCS2. This gives rise to the saving in communication overhead as follows

$$\frac{|hash(\cdot)| + 2|q| - (|KH.(\cdot)| + |q|)}{|hash(\cdot)| + 2|q|} = \frac{|q|}{\frac{1}{2}|q| + 2|q|} = 40\%$$

In conclusion, when compared with signature-then-encryption on elliptic curves, signcryption on the curves represents a 58% saving in computational cost and a 40% saving in communication overhead.

A.3 Security Assessment and Argument, Known Limitations, and Patent Issues

Similar to those for schemes over finite fields.

B Fast Computation of the Product of Multiple Exponentials with the Same Modulo

In unsigncryption, the most expensive part of computation is contributed by $g_0^{e_0} g_1^{e_1} \bmod p$, where g_0, g_1, e_0, e_1 and p are all large integers. Although the computation can be carried out in a straightforward way, namely computing $y_0 = g_0^{e_0} \bmod p$ and $y_1 = g_1^{e_1} \bmod p$ separately and then multiplying y_0 and y_1 together, it was observed by A. Shamir that as the product involves the same modulo, the final result can be obtained with a smaller computational cost by using a variant of the "square-and-multiply" method for exponentiation (see [21] as well as Algorithm 14.88 on Page 618 of [38]). The following algorithm embodies Shamir's technique to compute the product of k exponentials with the same modulo.

INPUT: Integers p, g_0, g_1, $\cdots$, g_{k-1} and e_0, e_1, $\cdots$, e_{k-1}, where the size of each e_i is t bits.

OUTPUT: $g_0^{e_0} g_1^{e_1} \cdots g_{k-1}^{e_{k-1}} \bmod p$.

1. Let E be a $k \times t$ binary array whose jth row is the binary representation of e_j, where $0 \leqq j \leqq k-1$. Call E an exponent array.
2. Denote by I_i the integer represented by the ith column of the exponent array E, with the least significant bit of I_i being the bit located at the 0th row in the ith column.
3. For i from 1 up to $2^k - 1$, pre-compute $G_i = \prod_{j=0}^{k-1} g_j^{i_j} \bmod p$, where $i = (i_{k-1} \cdots i_0)_2$.
4. set $A = 1$.
5. For i from 1 up to t,
   (a) Let $A = A \cdot A \bmod p$,
   (b) If $I_i \neq 0$, let $A = A \cdot G_{I_i} \bmod p$.
6. Return A as the final result.

Now we analyze the computational complexity of the algorithm. First we note that for a small k, say $k \leqq 4$, the computational cost for pre-computing G_1, G_2, $\ldots$, G_{2^k-1} is marginal when compared to the total cost for computing the product. In other words, the total computational cost is dominated by $(t+v)$ modulo multiplications invested in updating A, where v is the number of non-zero columns in the exponent array E. For e_0, e_1, $\cdots$, e_{k-1} chosen independently at random, one expects that $(\frac{1}{2})^k t$ of the columns in E are zeros. Thus the expected number of modulo multiplications is $(2 - (\frac{1}{2})^k)t$.

For $k = 2$, the expected computational cost is $1.75t$ modulo multiplications. This is roughly equivalent to 1.17 modulo exponentiations when the standard "square-and-multiply" method is used.

Example 1 Let $k = 2$ and $t = 6$. Assume that we are given the following two exponents: $e_0 = 60 = (111100)_2$, and $e_1 = 20 = (010100)_2$. Then the exponent array E is:

E =

	I_1	I_2	I_3	I_4	I_5	I_6
e_0	1	1	1	1	0	0
e_1	0	1	0	1	0	0

The pre-computation (Step 3) gives:

i	0	1	2	3
G_i	1	g_0	g_1	$g_0 g_1$

And iterations in Step 5 update A by way of:

i	1	2	3	4	5	6
A	g_0	$g_0^3 g_1$	$g_0^7 g_1^2$	$g_0^{15} g_1^5$	$g_0^{30} g_1^{10}$	$g_0^{60} g_1^{20}$

The total number of modulo multiplications in Step 5 is therefore $6 + 2 = 10$.