Signcryption

How to Achieve
Cost(signature & encryption) <<
Cost(signature) + Cost(encryption)

Yuliang Zheng
Calyptix Security Corporation, and
UNC Charlotte
www.signcryption.net

Applications of Signcryption

- Bring to society huge savings in comp. & comm. if used widely in
  - secure & authenticated message delivery / storage
  - electronic commerce
    - secure & authenticated transactions !!!
  - secure & authenticated multicast (incl. video conference, CSCW etc)
  - fast, compact, secure, unforgeable & non-repudiated key transport
  - ......
Goals

- To provide with
  - both confidentiality,
  - and authenticity
    - unforgeability &
    - non-repudiation
- but in an efficient way!

To hit 2 birds using 1 boomerang!
Outline

- problems with sign-then-encrypt
- signcryption -- a new paradigm
- cost-savings of signcryption over sign-then-encrypt
- properties of signcryption
  - confidentiality, unforgeability and non-repudiation
- signcryption for multiple recipients
- applications

In the paper & ink world:
Signature-then-Seal

To achieve: authenticity (unforgeability & non-repudiation)

To achieve: confidentiality
In the digital world (Alice to Bob):

**Signature-then-Encryption**

1. **Signature generation**
   - Alice signs a message $m$ using her secret key, i.e. creating $\text{sig}$ on $m$.

2. **Encryption**
   - Alice encrypts $(m, \text{sig})$ using DES with $k$.
   - Alice creates another data so that Bob can recover $k$. (Typically, Alice encrypts $k$ using Bob’s public key).
**Signature-then-Encryption (based on RSA)**

- EXP=2+2
- \( m \) encrypted using a private key cipher with \( k \)
- \( \text{sig} \)
- \( k^{e_b} \) encrypted with the receiver's public key \( e_b \)
- **comm. overhead**

**Signature-then-Encryption (based on Discrete Logarithm or DL)**

- EXP=3+2.17
- \( m \) encrypted using a private key cipher with \( k \)
- \( \text{sig} \)
- \( g^x \) used by the receiver to reconstruct \( k \)
- **comm. overhead**
Cost of Signature-then-Encryption

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Cost (No. of exp)</th>
<th>Comp Cost (bits)</th>
<th>Comm Overhead (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA based sig-then-enc</td>
<td>2 + 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL based Schnorr sig + ElGamal enc</td>
<td>3 + 2.17 (3 + 3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where hash is a 1-way hash function.

Why signature-then-encryption can be a problem

- Consider a transaction/message of 5,120 bits (=640 chars, ≈ 8 lines) that requires
  - high level security, or
  - to be transmitted in 2010
- Very large moduli, say of 5120 bits, have to be used
Why signature-then-encryption can be a problem (cnt’d)

- If RSA with a 5120-bit composite is used
  - **Comp. cost:**
    2+2=4 exponentiations mod a (very large !) 5120-bit integer
  - **Comm. overhead:**
    10,240 bits (twice as large as the original message !)

<table>
<thead>
<tr>
<th>5,120 bits</th>
<th>5,120 bits</th>
<th>5,120 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>sig</td>
<td>$k^{c_b}$</td>
</tr>
</tbody>
</table>

- If Schnorr sig & ElGamal enc with a 5120-bit prime are used
  - **Comp. cost:**
    3+2.17=5.17 (3+3=6) exponentiations mod a (very large !) 5120-bit integer
  - **Comm. overhead:**
    >= 5560 bits

<table>
<thead>
<tr>
<th>5,120 bits</th>
<th>5,120 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>sig</td>
</tr>
<tr>
<td>&gt;=5,560 bits</td>
<td>$g^x$</td>
</tr>
</tbody>
</table>
Signcryption -- a new approach

- Achieves the functions of
  - digital signature
    - unforgeability & non-repudiation
  - encryption
    - confidentiality
- has a **significantly smaller**
  comp. & comm. cost

\[
\text{Cost (signcryption)} << \text{Cost (signature)} + \text{Cost (encryption)}
\]
Signcryption --
public & secret parameters

- Public to all
  - \( p \): a large prime
  - \( q \): a large prime factor of \( p-1 \)
  - \( g \): \( 0 < g < p \) & with order \( q \) mod \( p \)
  - \textit{hash}: 1-way hash
  - \( \text{KH} \): key-ed 1-way hash
  - \((E,D)\):
    private-key encryption & decryption algorithms

- Alice’s keys
  - \( x_a \): secret key
  - \( y_a \): public key
  (note:
    \[ y_a = g^{x_a} \text{ mod } p \])

- Bob’s keys
  - \( x_b \): secret key
  - \( y_b \): public key
  (note:
    \[ y_b = g^{x_b} \text{ mod } p \])

---

Key-ed 1-way hash: examples

- efficient, but security properties are less understood

\[
KH_k(x) \equiv \text{hash}(k, x)
\]

where \( \text{hash} \) is a 1-way hash.
Key-ed 1-way hash: examples
(cnt’d)

- (slightly) less efficient, but some properties can be proven

- NMAC:
  \[ KH_{k_1,k_2}(x) = F_{k_1}(F_{k_2}(x)) \]
  where \( F_k(x) \) is the same as \( \text{hash} \), except that IV used by \( \text{hash} \) is now replaced by \( k \).

- HMAC:
  \[ KH_k(x) = \text{hash}(k' \oplus \text{opad}, \text{hash}(k' \oplus \text{ipad}, x)) \]
  where \( k' \) is a 0-padded version of \( k \), \( \text{opad} = \text{x36…36} \), \( \text{ipad} = \text{x5c…5c} \)

Signcrypition -- 1st example

- **Signcryption by Alice**
  - \( k = \text{hash}(y^x_b \mod p) \)
    where \( x \in \mathbb{Z}_{K+q-1} \)
  - \( k \rightarrow k_1 \rightarrow k_2 \)
  - \( r = KH_{k_2}(m) \)
  - \( s = \frac{x}{r + x_a} \mod q \)
  - \( c = E_{k_1}(m) \)
  - **output** \((c,r,s)\)

- **Unsigncryption by Bob**
  - \( k = \text{hash}((y_a \cdot g^r)^{x_a} \mod p) \)
  - \( k \rightarrow k_1 \rightarrow k_2 \)
  - \( m = D_{k_1}(c) \)
  - **output**
    \[ \begin{cases} 
    m & \text{if } r = KH_{k_2}(m) \\
    \text{"invalid"} & \text{if } r \neq KH_{k_2}(m) 
    \end{cases} \]
Signcryption -- 1st example (focusing on unsigncryption)

Let
\[ u = s \cdot x_b \mod q, \]
\[ v = r \cdot u \mod q, \]
Then,
\[ (y_a \cdot g^r)^{s \cdot x_b} \mod p = (y_a^{s \cdot x_b} \cdot g^{r \cdot s \cdot x_b}) \mod p = (y_a^u \cdot g^v) \mod p \]

This can be done in a smart way, costing only 1.17 exponentiations on average!

Signcryption -- 2nd example

\[ m \rightarrow (c,r,s) \]
\[ (c,r,s) \rightarrow m \]

- **Signcrypt by Alice**
  - \( k = \text{hash} \left( y_a^x \mod p \right) \)
  - where \( x \in \{1, K, q - 1\} \)
  - \( k \rightarrow k_1 \)
  - \( k \rightarrow k_2 \)
  - \( r = KH_{k_2}(m) \)

  - \( s = \frac{x}{1 + x_a \cdot r} \mod q \)
  - \( c = E_{k_1}(m) \)
  - output \( (c,r,s) \)

- **Unsigncrypt by Bob**
  - \( k = \text{hash}((g \cdot y_a^x)^{r \cdot s} \mod p) \)

  - \( k \rightarrow k_1 \)
  - \( k \rightarrow k_2 \)

  - \( m = D_{k_1}(c) \)

  - output

\[
\begin{align*}
 m & \quad \text{if } r = KH_{k_2}(m) \\
 "\text{invalid}" & \quad \text{if } r \neq KH_{k_2}(m)
\end{align*}
\]
Signcryption -- 3rd example

m → (c,r,s) (c,r,s) → m

- **Signcrypt by Alice**
  - $k = \text{hash}(y_b^x \mod p)$
  - where $x \in \{1, K, q - 1\}$
  - $k \leftarrow k_1 \leftarrow k_2$
  - $r = KH_{k_2}(m)$
  - $s = (x - r \cdot x_a) \mod q$
  - $c = E_{k_1}(m)$
  - output $(c,r,s)$

- **Unsigncrypt by Bob**
  - $k = \text{hash}((g^r \cdot y_a^s)^x \mod p)$
  - $k \leftarrow k_1 \leftarrow k_2$
  - $m = D_{k_1}(c)$
  - output
    - $m$ if $r = KH_{k_2}(m)$
    - "invalid" if $r \neq KH_{k_2}(m)$

Cost of Signcryption

**Alice**

m → (c,r,s)

**Bob**

(c,r,s) → m

- **Comp. cost**
  - by Alice:
    - 1 exponentiation modulo $p$
  - by Bob:
    - 1.17 exponentiations modulo $p$ (using Shamir’s technique)
  - total = 2.17 exp mod $p$

- **Comm. overhead**
  - $|r| + |s|$ bits
  - (note: $|m| = |c|$)
**Comp. & Comm. Cost of Signcryption**
*(based on Discrete Logarithm)*

**EXP=1+1.17**

- encrypted using a private key cipher with $k$
- comm. overhead

---

**“Magic” Signcryption Envelope**

---

(C) 1997+ Yuliang Zheng
Necessity of Binding a Recipient’s Name

- Some schemes, s.a. SCS1 and SCS2, may have a problem with “double-payment”:

- if \( x_b \) and \( x_c \) are related, say by
  \[
  x_b = w \cdot x_c \mod q
  \]

- Why?
  \[\text{If } (c, r, s) \text{ is from Alice to Bob, then is a valid msg from Alice to Cathy!}\]

How to Tie a Recipient’s name to a Signcrypted Text?

- Simply include data on Bob’s ID, s.a. his public key/certificate, into the computation of \( r \),

- Namely, change \( r \)
  
  from \( r = KH_{k_2}(m) \)
  
  to \( r = KH_{k_2}(m, y_b, etc) \)
signature-then-encryption
v.s.
signcryption

Signcryption v.s. Signature-then-Encryption

(a) Signcryption based on DL
(b) Signature-then-Encryption based on RSA
(c) Signature-then-Encryption based on DL
Cost of Signature-then-Encryption v.s. Cost of Signcryption

A simplistic comparison:

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Cost (No. of exp)</th>
<th>Comp Cost</th>
<th>Comm Overhead (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA based sig-then-enc</td>
<td>2 + 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL based Schnorr sig + ElGamal enc</td>
<td>3 + 2.17 (3 + 3)</td>
<td>1+ 1.17 (1 + 2)</td>
<td></td>
</tr>
</tbody>
</table>

A more detailed comparison

- **Why do this?**
  - the computing time of \( y^x \mod z \) largely depends on the size of \( x \)
  - the sizes of RSA & DL exponents are different
    - DL exponent --- [1,...,q ]
    - RSA public exponent \( e \) --- can be small
    - RSA secret exponent \( d \) --- |n| bits, BIIIGG!
Signcryption v.s. Schnorr Sig + ElGamal Enc

- Saving in comp. cost by signcryption
  \[ \frac{(5.17 - 2.17) \mod \text{exp}}{5.17 \mod \text{exp}} = 58\% \]

- Saving in comm. overhead by signcryption (assuming |hash|=|KH|)
  \[ \frac{|p|}{|KH(\cdot)| + |q| + |p|} \]

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>KH</th>
<th>saving in comp cost</th>
<th>saving in comm overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>144</td>
<td>72</td>
<td>58 %</td>
<td>70.3 %</td>
</tr>
<tr>
<td>768</td>
<td>152</td>
<td>80</td>
<td>58 %</td>
<td>76.8 %</td>
</tr>
<tr>
<td>1024</td>
<td>160</td>
<td>80</td>
<td>58 %</td>
<td>81.0 %</td>
</tr>
<tr>
<td>1536</td>
<td>176</td>
<td>88</td>
<td>58 %</td>
<td>85.3 %</td>
</tr>
<tr>
<td>2048</td>
<td>192</td>
<td>96</td>
<td>58 %</td>
<td>87.7 %</td>
</tr>
<tr>
<td>3072</td>
<td>224</td>
<td>112</td>
<td>58 %</td>
<td>90.1 %</td>
</tr>
<tr>
<td>4096</td>
<td>256</td>
<td>128</td>
<td>58 %</td>
<td>91.0 %</td>
</tr>
<tr>
<td>5120</td>
<td>288</td>
<td>144</td>
<td>58 %</td>
<td>92.0 %</td>
</tr>
<tr>
<td>8192</td>
<td>320</td>
<td>160</td>
<td>58 %</td>
<td>94.0 %</td>
</tr>
<tr>
<td>10240</td>
<td>320</td>
<td>160</td>
<td>58 %</td>
<td>96.0 %</td>
</tr>
</tbody>
</table>
Signcryption v.s. RSA

- Adv. in comp. cost by signcryption
  \[ \frac{0.375(|n_a| + |n_b|) - 3.25q}{0.375(|n_a| + |n_b|)} \]
  (Assuming small RSA public exponents & CRT are used!)

- Adv. in comm. overhead by signcryption
  \[ \frac{|n_a| + |n_b| - [KH(\cdot) + q]}{|n_a| + |n_b|} \]

<table>
<thead>
<tr>
<th>p</th>
<th>n_a</th>
<th>n_b</th>
<th>q</th>
<th>KH</th>
<th>saving in comp cost</th>
<th>saving in comm overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>144</td>
<td>72</td>
<td>0</td>
<td>72</td>
<td>0 %</td>
<td>78.9 %</td>
</tr>
<tr>
<td>768</td>
<td>152</td>
<td>80</td>
<td>1.42</td>
<td>80</td>
<td>14.2 %</td>
<td>84.9 %</td>
</tr>
<tr>
<td>1024</td>
<td>160</td>
<td>80</td>
<td>3.23</td>
<td>80</td>
<td>32.3 %</td>
<td>88.3 %</td>
</tr>
<tr>
<td>1536</td>
<td>176</td>
<td>88</td>
<td>5.03</td>
<td>88</td>
<td>50.3 %</td>
<td>91.4 %</td>
</tr>
<tr>
<td>2048</td>
<td>192</td>
<td>96</td>
<td>5.94</td>
<td>96</td>
<td>59.4 %</td>
<td>93.0 %</td>
</tr>
<tr>
<td>3072</td>
<td>224</td>
<td>112</td>
<td>6.84</td>
<td>112</td>
<td>68.4 %</td>
<td>94.0 %</td>
</tr>
<tr>
<td>4096</td>
<td>256</td>
<td>128</td>
<td>7.29</td>
<td>128</td>
<td>72.9 %</td>
<td>95.0 %</td>
</tr>
<tr>
<td>5120</td>
<td>288</td>
<td>144</td>
<td>7.56</td>
<td>144</td>
<td>75.6 %</td>
<td>96.0 %</td>
</tr>
<tr>
<td>8192</td>
<td>320</td>
<td>160</td>
<td>8.31</td>
<td>160</td>
<td>83.1 %</td>
<td>97.0 %</td>
</tr>
<tr>
<td>10240</td>
<td>320</td>
<td>160</td>
<td>8.65</td>
<td>160</td>
<td>86.5 %</td>
<td>98.0 %</td>
</tr>
</tbody>
</table>
DL Signcryption v.s. sign-then-encrypt

# of multiplications

![Graph showing the comparison of multiplications for RSA sign-enc, Schnorr + ElGamal, and DL Signcryption.](image)

DL Signcryption v.s. sign-then-encrypt

comm. overhead

(# of bits)

![Graph showing the comparison of communication overhead for RSA sign-enc, Schnorr + ElGamal, and DL Signcryption.](image)
DL Signcryption v.s.
RSA sign-encrypt

# of multiplications

| $|p| = |n|$ |
|---|---|
| 1024 | 2048 | 4096 | 8190 |
| RSA sign-enc | 0 | 1000 | 2000 | 3000 |
| DL Signcryption | 1000 | 2000 | 3000 | 4000 |

DL Signcryption v.s.
RSA sign-encrypt

comm. overhead

($\#$ of bits)

| $|p| = |n|$ |
|---|---|
| 1024 | 2048 | 4096 | 8190 |
| RSA sign-enc | 0 | 5000 | 10000 | 15000 |
| DL Signcryption | 5000 | 10000 | 15000 | 20000 |
Why Can Signcryption Save Cost?

- For Bob to successfully recover a message, he needs to know $g^x \mod p$ from which he can compute $k = \text{hash}(g^{x-a} \mod p)$.

- With signcryption, Bob can derive $g^x \mod p$ from $r, s, g, p$ and $y_a$. That is, there is NO need for Alice to explicitly send Bob $g^x \mod p$.

A general method for implementing signcryption.
Signcryption can be based on any shortened ElGamal-like signatures

How to shorten ElGamal-like signatures:

- **Calculation of $r$**
  - $r = \text{hash}(k,m)$
  - where $k = g^x \mod p$
  - and $x$ is a random number.

- **Calculation of $s$**
  - if $\text{hash}(m)$ is in the original $s$:
    - $\text{hash}(m) \rightarrow 1$
    - OR
    - $r \rightarrow 1$, $\text{hash}(m) \rightarrow r$
  - otherwise if $\text{hash}(m)$ is not in the original $s$, go to next step.
  - if $s = (...) / x$, then change it to $s = x / (...)$. 

### Shortened DSS (for Alice)

<table>
<thead>
<tr>
<th>$m$</th>
<th>$(m, r, s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>$(m, r, s)$</td>
</tr>
</tbody>
</table>

- **Shortened DSS -- type 1**
  - $k = g^x \mod p$
  - where $x \in \{1, K, q - 1\}$
  - $r = \text{hash}(k,m)$
  - $s = \frac{x}{r + x_a} \mod q$
  - output $(m, r, s)$

- **Shortened DSS -- type 2**
  - $k = g^x \mod p$
  - where $x \in \{1, K, q - 1\}$
  - $r = \text{hash}(k,m)$
  - $s = \frac{x}{1 + x_a \cdot r} \mod q$
  - output $(m, r, s)$
A generic signcryption scheme based on any shortened signature

**Signcrypt by Alice**
- $k = \text{hash}(y_b^x \mod p)$
  - where $x \in \mathbb{Z}_p \{1, K, q - 1\}$
- $k \overset{\leftarrow}{\rightarrow} k_1$
- $r = KH_{k_2}(m)$
- $s = s(x, r, x_u, q, K)$
- $c = E_{k_1}(m)$
- output $(c, r, s)$

**Unsigncrypt by Bob**
- $k = \text{hash}(k(g, r, s, x_b, y_a, p, K))$
- $k \overset{\leftarrow}{\rightarrow} k_1$
- $m = D_{k_1}(c)$
- output
  - $m$ if $r = KH_{k_2}(m)$
  - "invalid" if $r \neq KH_{k_2}(m)$

unforgeability
non-repudiation
&
confidentiality
Unforgeability of Signcryption

- No body, even Bob the recipient, can forge a valid & signcryted text from Alice

- How to prove it?
  - in the random oracle model
  - use Pointcheval & Stern’s Eurocrypt96 proof technique

Non-repudiation of Signcryption

- With signature-then-encryption, the origin of a decrypted message can be universally verified.

- The origin of an unsigncrypted message, however, can be **directly** verified only by Bob the recipient (using his secret key). It does NOT satisfy “universal verifiability”.
Non-repudiation of Signcryption (cnt’d)

- “Direct verifiability by the recipient only” is exactly what Alice and Bob want in normal situations!
- It is also a main reason (aside from security consideration) why traditionally one uses “signature-then-encryption” rather than “encryption-then-signature”!!

Non-repudiation of Signcryption (cnt’d)

- Alice cannot deny the fact that she is the originator of a signcrypted text.
- When requested by Bob, a judge can settle a dispute through interactions with Bob.
Repudiation Settlement Methods

- Simple if
  - a trusted tamper-resistant device is used, or
  - the judge is 100% trusted
- Using a ZK interactive protocol (s.a. Bellare-Jakobsson-Yung Protocol presented at Eurocrypt97) if Bob does not trust the judge
  - Bob “guides” the judge to verify the origin of a message, without revealing his private key $x_b$

Circuit for ZK Interactive Repudiation Settlement
Repudiation Settlement for Signature-then-Encryption

1 move of data

Repudiation Settlement for Signcryption

4 move ZK protocol
Confidentiality of Signcryption

- A third party cannot obtain information on a message $m$ sealed by a signcryption scheme, if all the underlying primitives are secure (incl: $<E,D>$, KH, DH, etc)

Confidentiality of Signcryption (cnt’d)

- How to prove it?
  - an attacker for a signcryption scheme
    
    $m \rightarrow (c, r, s)$

    can be translated into one for a secure encryption scheme defined by
    
    $m \rightarrow (c, u, r)$

    where
    
    $c = E_{k_1}(m)$, $u = g^s \mod p$, $r = KH_{k_2}(m)$

    $k_1 || k_2 = \text{hash}(y_b^s \mod p)$
Other aspects of signcryption v.s. sign-then-enc

<table>
<thead>
<tr>
<th>Attribute paradigm</th>
<th>forward secrecy w.r.t. Alice</th>
<th>past recovery</th>
<th>static key manage.</th>
<th>Repudi. Settle.</th>
<th>“group” orient.</th>
</tr>
</thead>
<tbody>
<tr>
<td>signcryption</td>
<td>no</td>
<td>yes</td>
<td>n/a</td>
<td>Interactive</td>
<td>yes</td>
</tr>
<tr>
<td>sign-then-enc</td>
<td>yes (but, forgeable)</td>
<td>no</td>
<td>n/a</td>
<td>non-interactive</td>
<td>no</td>
</tr>
<tr>
<td>sign-then-enc with a static key</td>
<td>no</td>
<td>yes</td>
<td>distrib/derivation/storage</td>
<td>non-interactive</td>
<td>yes (in most cases)</td>
</tr>
</tbody>
</table>

signcryption for multiple recipients
Signature-then-Encryption for Multi-Recipients (RFC1421, RSA)

\[ \text{EXP} = (t+1) + 2t \]

- \( m \) encrypted using a private key cipher with \( k \)
- \( k^{e_1} \) encrypted with the receiver \( R_1 \)'s public key \( e_1 \)
- \( k^{e_t} \) encrypted with the receiver \( R_t \)'s public key \( e_t \)

\[ \text{EXP} = (t+1) + 2t \]

Signature-then-Encryption for Multi-Recipients (based on DL)

\[ \text{EXP} = (2t+1) + 2.17t \]

- \( m \) encrypted using a private key cipher with \( k \)
- \( g^{x_1} \) encrypted using a private key cipher with \( k_j \) used by the receiver \( R_j \) to reconstruct \( k_j \)
- \( g^{x_t} \) encrypted using a private key cipher with \( k_t \) used by the receiver \( R_t \) to reconstruct \( k_t \)
Signcryption for Multi-Recipients
(based on DL)

EXP = t + 1.17t

- m
- KH(m)
- sig

encrypted using a private key cipher with k
encrypted using a private key cipher with k_{i,j}
encrypted using a private key cipher with k_{t,1}

Signcryption for multiple recipients --
public & secret parameters

- Public to all
  - p: a large prime
  - q: a large prime factor of p-1
  - g: 0 < g < p & with order q mod p
  - hash: 1-way hash
  - KH: key-ed 1-way hash
  - (E,D): private-key encryption & decryption algorithms

- Alice’s keys
  - x_a: secret key
  - y_a: public key
    (note: \( y_a = g^{x_a} \mod p \))

- Recipient R_i’s keys
  - x_i: secret key
  - y_i: public key
    (note: \( y_i = g^{x_i} \mod p \))

(C) 1997+ Yuliang Zheng
Signcryption by Alice for recipients $R_1,...,R_t$

$c = E_k(m \parallel h)$, where $h = KH_k(m)$ and $k$ is a random key

- for $i = 1,...,t$
  
  $k_i = \text{hash}(y_i^{v_i} \mod p)$ with $v_i \in_R \{1,K,q-1\}$

  \[ k_i \rightarrow k_{i,1} \]
  \[ k_{i,2} \]

  $c_i = E_{k_{i,1}}(k)$

  $r_i = KH_{k_{i,2}}(m,h)$

  $s_i = v_i / (r_i + x_q) \mod q$

- broadcast $(c,c_1,r_1,s_1,K,c_i,r_i,s_i)$

Unsigncryption by each recipient $R_i$, $i=1,...,t$

- find out $(c,c_i,r_i,s_i)$ in $(c,d_1,r_1,s_1,K,c_i,r_i,s_i)$

  $k_i = \text{hash}(((y_a \cdot g^{r_i})^{s_i \cdot x_i}) \mod p)$

  \[ k_i \rightarrow k_{i,1} \]
  \[ k_{i,2} \]

  \[ k = D_{k_{i,1}}(c_i) \]

- $w = D_k(c)$

  \[ w \rightarrow m \]
  \[ h \]

- output \( \begin{cases} m & \text{if } h = KH_k(m) \text{ and } r_i = KH_{k_{i,2}}(w) \\ "invalid" & \text{otherwise} \end{cases} \)
Signcryption v.s. Signature-then-Encryption for Multi-Recipients

(a) Signcryption based on DL
(b) Signature-then-Encryption based on RSA
(c) Signature-then-Encryption based on DL

Cost-saving of signcryption for \( t \) recipients

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Cost (no. of exp.)</th>
<th>comp. Cost</th>
<th>comm. overhead (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schnorr signature + ElGamal encryption</td>
<td>Alice: 2t + 1</td>
<td>2.17</td>
<td>( t \cdot (</td>
</tr>
<tr>
<td>RFC1421 (RSA)</td>
<td>Alice: t + 1</td>
<td>2</td>
<td>(</td>
</tr>
<tr>
<td>signcryption</td>
<td>Alice: t</td>
<td>1.17</td>
<td>( t \cdot (</td>
</tr>
</tbody>
</table>

(C) 1997+ Yuliang Zheng
Applications of Signcryption

- Bring to society huge savings in comp. & comm. if used widely in
  - secure & authenticated message delivery / storage
  - electronic commerce
    - secure & authenticated transactions !!!
  - secure & authenticated multicast (incl. video conference, CSCW etc)
  - fast, compact, secure, unforgeable & non-repudiated key transport
Secure and authenticated key transport in a single ATM cell

ATM Cell

5 bytes 48 bytes (384 bits)

header payload (data)

<table>
<thead>
<tr>
<th>c</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>144 bits</td>
<td>80 bits</td>
<td>160 bits</td>
</tr>
</tbody>
</table>

$|p| \geq 512, \ |q| \geq 160, \ |KH(\cdot)| \geq 80$

$(k_1,k_2) = \text{hash}(y_b^* \mod p)$ with $x \in [1,K,q-1]$

$|k_1| \geq 64, \ |k_2| \geq 64$

$c = E_{k_1}(\text{key}, \text{TQ})$

$r = KH_{k_2}(\text{key}, \text{TQ}, \text{other})$

$s = \frac{x}{r+x_a} \mod q$

Signcryption cannot be achieved with a shared secret key alone

- Alice and Bob may use a shared secret key, such as $g^{x_a \cdot x_b} \mod p$ or a KPS key, to carry out secure and efficient communications with content integrity.

- But, without a tamper-proof device and/or a trusted 3rd party, such communications may not achieve unforgeability or non-repudiation.
Extensions

- **Signcryption** can be built on other versions of the discrete logarithm, such as those on **elliptic curves**.
- Lenstra’s new sub-groups presented at ACISP’97 can also be used in signcryption.

Stop Press, May 2000

- We have recently succeeded in implementing signcryption schemes based on factoring RSA moduli (details are available upon request).
- Thus, signcryption can now be based on
  - Discrete Log on finite fields
  - Elliptic curves
  - Factoring
Summary

- introduced "digital signcryption" that achieves
  - confidentiality
  - unforgeability & non-repudiation
- proposed concrete implementations
- analysed the significant saving of signcryption over "signature-then-encryption"