RSA II

RSA signature, use in PGP, and RSA CRT

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

Schoolbook RSA signatures

1977 Rivest, Shamir, Adleman. Do not use Schoolbook RSA in practice!

Schoolbook RSA signatures

1977 Rivest, Shamir, Adleman. Do not use Schoolbook RSA in practice!

KeyGen:

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

Schoolbook RSA signatures

1977 Rivest, Shamir, Adleman. Do not use Schoolbook RSA in practice!

KeyGen:

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

Sign message m with $0 \leq h(m) < n$:

1. Compute $s \equiv (h(m))^d \bmod n$.
2. Output s.

Verify signature $0 \leq c < n$:

1. Compute $h' \equiv s^e \bmod n$.
2. Output true if $h' = h(m)$. Else output false.

Schoolbook RSA signatures

1977 Rivest, Shamir, Adleman. Do not use Schoolbook RSA in practice!

KeyGen:

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

Sign message m with $0 \leq h(m) < n$:

1. Compute $s \equiv (h(m))^d \bmod n$.
2. Output s.

Verify signature $0 \leq c < n$:

1. Compute $h' \equiv s^e \bmod n$.
2. Output true if $h' = h(m)$. Else output false.

This works for the same reason as RSA encryption works.
We use $h(m)$ to sign arbitrarily long messages.

Schoolbook RSA signatures

1977 Rivest, Shamir, Adleman. Do not use Schoolbook RSA in practice!

KeyGen:

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

Sign message m with $0 \leq h(m) < n$:

1. Compute $s \equiv (h(m))^d \bmod n$.
2. Output s.

Verify signature $0 \leq c < n$:

1. Compute $h' \equiv s^e \bmod n$.
2. Output true if $h' = h(m)$. Else output false.

This works for the same reason as RSA encryption works.
We use $h(m)$ to sign arbitrarily long messages.

Note that RSA is unusual in that signing matches decryption.

What does RSA use in practice look like?

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBGFdN9UBCADBI4YB18EGJiotgAzKwcPGFbo5f+QK2kT7LPOO+8aHF1zRAO/w
1hLymwN9uMU+/fglIaSFIuzo3kniZafukKAuy3cx5DJWRs/Ec4T2nQStICe2H6Mu
7MjoSkakQnX11bMBV94Px8ILBL0bqs9h9W0s4KkeBPTRA1899WsnSXUCK1f1s9rv
XRheQVkT77QoogdOhbAJEIbEswB/mICO9RX9QxxOp2wY7lEBc4hQIsjRL0RQr6uv
TRUZzVjSujZHkXR/ZzaV1N6RBtN55s6doHgnYr1ueYMfCXPcBZFdRdSago2gVdgo
xJTgQrNF8UUAY9Eodw2hKXS/PLeFYNyUTtCtABEBAAG0QER1bW15IEtleSAoZHVt
bXkga2V5IGdlbmVyYXRlZCBmb3IgdGVhY2hpbmcpIDxkdW1teUBleGFtcGxlLmNv
bT6JATgEEwECACIFAmFdN9UCGwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJ
EDknRU2TvDpAKVcIALm4mDccRt0n6CV+S74191r7pizzF5BWKdEtiI7SnEjpSS98
aBVSQRdXtxEsGOVb7DB8VJS/nLIs30oFWqJb4EXAKVoG3fAbFVryEjkEv9FzwQDb
FkZXC/hsQ+pGntZ3frzUYQ3VLTn+7hZcATyP5a6NkjkxMahhCaGmyddTOrdVqHK5
ChtenI2j51zc032SJABcqQY1C2qjmfNoTfdB/m8QxSoHiW+5S9g35JvOXYcQ+Mjy
WA/lC6dkTkosAybnQLw+DUn+x08WXHtuxj0sgJ52/dN8iKoQIyhZtB+mM2LBcj6A
+PTlR3pOXpBBs1GAP6JbE7vNw+VdDRXU4TlKuBq5AQ0EYV031QEIAN81N3ytxoWg
OFB2opxC+WhpOADZOsrCOibUTjTlz2OMEj2mfreGAqr3qmxM1MMXeB0jptcUoFka
ab+594hntwqSCU7FxCzHPd93bZFXXhIWHhQ2OIEwOaWa7MXFqg8sYV7ef3AWX8wq
DP3cgeek4yf/ITQZqV2bX8IdIFFnDYEuL7XwUrsdcKgois7/SA+L4HRqQnDlskBs
/ulEjA3Ae5YaoFBc4hO+fGzp5u8EbJ29FVRzvJ40OYqSD5rk9vriuIQF74T5/tY6
3XtF6JWHcty0laqXpB1uiBeAq6jKkuI6JpKxqf9C89bVpSscxL/slBX7Wr5Hu+Iq
jU22poCw5YMAEQEAAYkBHwQYAQIACQUCYV031QIbDAAKCRA5J0VNk7w6QNQaB/4v
SjAQtAtjqrWbKq7GfIHaJ8MN4qwmbfBl6pWw/rFxb8WZ8uzPmobK+WzqEFP8BbkM
K+BZzkQP93nFbauT0EjWXEZAQ0SeVwrxMMDeYvww8DS15wxKiZXwEw6QAq0hcg9E
GCO8ctmvpG59HmrqLkmxdEU4QZEEspYGUwtOEwysqspVwXPnbQlxWENqXRzl2pD/
```

Inspecting the key with pgpdump -i dummy-pub.asc

```
Old: Public Key Packet(tag 6)(269 bytes)
Ver 4 - new
Public key creation time - Wed Oct  6 07:44:53 CEST 2021
Pub alg - RSA Encrypt or Sign(pub 1)
RSA n(2048 bits) - c1 23 86 01 d7 c1 06 26 2a 2d 80 0c ca c1 c3 c6 15 ba 39 7f
RSA e(17 bits) - 01 00 01
Old: User ID Packet(tag 13)(64 bytes)
User ID - Dummy Key (dummy key generated for teaching) <dummy@example.com>
Old: Signature Packet(tag 2)(312 bytes)
Ver 4 - new
Sig type - Positive certification of a User ID and Public Key packet(0x13).
Pub alg - RSA Encrypt or Sign(pub 1)
Hash alg - SHA1(hash 2)
Hashed Sub: signature creation time(sub 2)(4 bytes)
Time - Wed Oct  6 07:44:53 CEST 2021
Hashed Sub: key flags(sub 27)(1 bytes)
Flag - This key may be used to certify other keys
Flag - This key may be used to sign data
Hashed Sub: preferred symmetric algorithms(sub 11)(5 bytes)
Sym alg - AES with 256-bit key(sym 9)
Sym alg - AES with 192-bit key(sym 8)
Sym alg - AES with 128-bit key(sym 7)
Sym alg - CAST5(sym 3)
Sym alg - Triple-DES(sym 2)
Hashed Sub: preferred hash algorithms(sub 21)(5 bytes)
Hash alg - SHA256(hash 8)
```

Inspecting the key with pgpdump -i dummy-sec.asc

```
Old: Secret Key Packet(tag 5)(920 bytes)
Ver 4 - new
Public key creation time - Wed Oct  6 07:44:53 CEST 2021
Pub alg - RSA Encrypt or Sign(pub 1)
RSA n(2048 bits) - c1 23 86 01 d7 c1 06 26 2a 2d 80 0c ca c1 c3 c6 15 ba 39 7f
RSA e(17 bits) - 01 00 01
RSA d(2047 bits) - 59 5e 4b a2 cc a7 c7 65 9f 7c a0 54 ca f9 2f d2 97 b9 2c e4
RSA p(1024 bits) - c8 11 a9 e7 60 59 58 2e ef d9 f0 9d 5c c2 ce d2 20 52 f7 3e
RSA q(1024 bits) - f7 21 e2 39 b7 07 45 24 4f f4 6d 36 9e 2f b6 8e 1b c4 73 d2
RSA u(1020 bits) - 0e 60 bb e0 23 91 c3 92 63 21 aa 8a e4 77 d0 f0 00 df 2b ea
Checksum - 3f 0b
Old: User ID Packet(tag 13)(64 bytes)
User ID - Dummy Key (dummy key generated for teaching) <dummy@example.com>
Old: Signature Packet(tag 2)(312 bytes)
Ver 4 - new
Sig type - Positive certification of a User ID and Public Key packet(0x13).
Pub alg - RSA Encrypt or Sign(pub 1)
Hash alg - SHA1(hash 2)
Hashed Sub: signature creation time(sub 2)(4 bytes)
Time - Wed Oct  6 07:44:53 CEST 2021
Hashed Sub: key flags(sub 27)(1 bytes)
Flag - This key may be used to certify other keys
Flag - This key may be used to sign data
Hashed Sub: preferred symmetric algorithms(sub 11)(5 bytes)
Sym alg - AES with 256-bit key(sym 9)
Sym alg - AES with 192-bit key(sym 8)
```

RSA CRT

RSA private key includes p, q, and u. This violates rule of forgetting pieces that are no longer used –

RSA CRT

RSA private key includes p, q, and u. This violates rule of forgetting pieces that are no longer used – but actually they are used.

RSA CRT uses the Chinese remainder theorem to speed up decryption/signing. Public exponent e can be chosen small but d has ℓ bit for n having ℓ bits.

$$m \equiv c^d \bmod n$$

does one exponentiation with ℓ-bit exponent.

Let $c_p \equiv c \bmod p$, $c_q \equiv c \bmod q$, $d_p \equiv d \bmod p-1$, and $d_q \equiv d \bmod q-1$.

$$m_p \equiv c_p^{d_p} \bmod p, \quad m_q \equiv c_q^{d_q} \bmod q$$

does two exponentiations with $\ell/2$-bit exponents each.

RSA CRT

RSA private key includes p, q, and u. This violates rule of forgetting pieces that are no longer used – but actually they are used.

RSA CRT uses the Chinese remainder theorem to speed up decryption/signing. Public exponent e can be chosen small but d has ℓ bit for n having ℓ bits.

$$m \equiv c^d \bmod n$$

does one exponentiation with ℓ-bit exponent.

Let $c_p \equiv c \bmod p$, $c_q \equiv c \bmod q$, $d_p \equiv d \bmod p-1$, and $d_q \equiv d \bmod q-1$.

$$m_p \equiv c_p^{d_p} \bmod p, \quad m_q \equiv c_q^{d_q} \bmod q$$

does two exponentiations with $\ell/2$-bit exponents each.
These are on half-size inputs and moduli. Each 2–4 times faster.

Combine m_p, m_q using CRT to

$$m = m_p + pu(m_q - m_p) \bmod n,$$

where $u \equiv p^{-1} \bmod q$, where $p < q$ (for uniqueness).