Comp 4104  RSA Encryption, Signatures

RSA

\[ n = n = pq \quad p, q = \text{safe primes} \]
\[ = (2p + 1)(2q + 1) \]

Choose element \( a \in \mathbb{Z}_n \) [0, n-1]

\( a \) is invertible if \( g \text{cd}(a, n) = 1 \)

Set of invertible elements is \( \mathbb{Z}_n^* \)

\[ \text{Ord}(a) | (p-1)(q-1) = \text{Ord}(a) | 4p'q' \]

In general, \( \text{Ord}(a) \mod b = \text{ord}(a) | \Phi(b) \)

\( \Phi(b) \) is the number of integers less than \( b \)
and co-prime to \( b \)

Find \( x \) given \( y \) & \( y \)

\[ y = xy \mod n \] easy to find if \( y \) is invertible
\[ x = yx \mod n \] \( y \in \mathbb{Z}_n^* \)

\[ y = y^x \mod n \rightarrow \text{Discrete Log} \rightarrow \text{still hard} \]
\[ x = y^x \mod n \rightarrow \text{Discrete roots} \rightarrow \text{hard} \]

Given \( n \), find \( p \) and \( q \) \rightarrow \text{Factoring} \rightarrow \text{hard}

\[ n \rightarrow \text{large } n \rightarrow |n| \geq 2048 \]
\( p \) and \( q \) are same length

\( b \leq (2^{128}) \) (NIST)

"Textbook" RSA Encryption

Key generation:

1) Pick two large safe primes \( p \) and \( q \)
2) Compute \( n = p \cdot q \)
3) pick integer $e$ (from $\mathbb{Z}_n^*$) → public → relatively prime to $(p-1)(q-1)$
4) compute $d=e^{-1} \mod (p-1)(q-1)$ → private → $ed \equiv 1 \mod (p-1)(q-1)$
5) Public key: $n$, $e$ (Private: $d$) by NIST

Encryption:
\[ c = \text{Enc}_{pk}(m) = m^e \mod n \]

Decryption:
\[ m = c^d \mod n = (m^e)^d \mod n \]

Textbook RSA:
1) Not randomized → encrypt same message, get same ciphertext
   → Not CPA-secure (nor CCA-secure)
2) Multiplicatively Homomorphic
   \[ \text{Enc}(m_1) \cdot \text{Enc}(m_2) = \text{Enc}(m_1 m_2) \]
   \[ = m_1^e m_2^e \mod n \]
   \[ = (m_1 m_2)^e \mod n \]

Signatures:
Recall 2 methods:
1) ZKP that you know secret key
   bound to a message
2) Sign by encrypting
   → doesn’t work for Elgamal because of randomization
   → works with "textbook" RSA
RSA Signature

\[ S = \text{Sign}_{sk}(m) = \text{Dec}_{sk}(m) \]
\[ \text{output } <m_S> = m^d \mod N \]

\[ \text{Verify}_{pk}(m_S) : S^e \mod n = m \]
\[ = m^e \mod n \]
\[ = m' \mod n \mod n \]
\[ = m \]

Union valid sig <m_S>

compute forgery as

\[ m' = m \cdot 2^e \mod n \]
\[ s' = 2s \mod n \]

Verify \( s', m' \):
\[ (s')^e \mod n = m' \mod n \]
\[ (2s)^e \mod n = m^2 \mod n \]
\[ 2^e (m')^e = m^2 \]
\[ 2^e m = 2^e m \]

Strengthening RSA Encryption/Signatures

- use padding scheme
- padding scheme is randomized
- padding scheme uses hash functions (assumed random oracle)
- \( \text{Enc}(m || \text{pad}(r)) \rightarrow \text{OAEP} \rightarrow \text{optimal asymmetric encryption padding} \)
- \( \text{Sig}(m || \text{pad}(r)) \rightarrow \text{DSA} \rightarrow \text{Probabilistic Signature Scheme} \)
- As secure as Schnorr or DSA
Encryption:  | Security:  | Signature:  | Security:  
---|---|---|---
Textbook RSA | OSS | "Textbook" RSA | Not secure
El Gamal | CPA | (D)DSA-DSS | secure
Twisted El Gamal | CPA | (D)DSA (DSS) | secure
(RSA-DAEP) | PKCS variant | Schnorr | secure

Key establishment:
1) Diffie-Hellman (textbook)  
   not secure, MITM
2) SSH w/ signatures  
   secure + never-given-read
3) Encryption-based (key transport)  
   secure

S/LS (sketch):

\[ A \leftarrow c^k \rightarrow B \]
\[ a \rightarrow y^a \rightarrow b \]
\[ K = g^{ab} \]
\[ g^b, \text{sig}(g^a, g^b) \]

Key transport:

\[ A \leftarrow pk_B \rightarrow B \]
\[ K \leftarrow \text{Enc}_{pk_B}(k) \rightarrow K \]

Full encrypted protocol:

1) Get Google's public key = pk
2) K = Key establishment (pk)
   \[ \text{Shared secret key} \]
3) k_enc, k_mac = PRG(K)
4) Enc_{k_E} (m || MAC_{k_M} || Pad)

Google

\[ \rightarrow \text{you want to connect to Google securely} \]

Primitives used:

| R8A, S8A | MDS |
| SHA256 |

Enc: RC4, AES-CTR
MAC: HMAC

\[ \rightarrow \text{MDS, SHA} \]