Public-key Cryptography
Theory and Practice

Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Chapter 5: Cryptographic Algorithms

Common Encryption Algorithms

Encryption algorithm	Security depends on
RSA encryption	Integer factoring problem
ElGamal encryption	DHP (DLP)
Rabin encryption	Square-root problem
Goldwasser-Micali encryption	Quadratic residuosity problem
Blum-Goldwasser encryption	Square-root problem
Chor-Rivest encryption	Subset sum problem
XTR	DLP
NTRU	Closest vector problem in lattices

RSA Encryption

- **Key generation**
  The recipient generates two random large primes p, q, computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, finds a random integer e with $\gcd(e, \phi(n)) = 1$, and determines an integer d with $ed \equiv 1 \pmod{\phi(n)}$.
  Public key: (n, e).
  Private key: (n, d).
- **Encryption**
  Input: Plaintext $m \in \mathbb{Z}_n$ and the recipient's public key (n, e).
  Output: Ciphertext $c \equiv m^e \pmod{n}$.
- **Decryption**
  Input: Ciphertext c and the recipient's private key (n, d).
  Output: Plaintext $m \equiv c^d \pmod{n}$.

RSA Encryption: Example

- Let $p = 257$, $q = 331$, so that $n = pq = 85067$ and $\phi(n) = (p-1)(q-1) = 84480$. Take $e = 7$, so that $d \equiv e^{-1} \equiv 60343 \pmod{\phi(n)}$.
  - Public key: $(85067, 7)$.
  - Private key: $(85067, 60343)$.
- Let $m = 34152$. Then $c \equiv m^e \equiv (34152)^7 \equiv 53384 \pmod{n}$.
- Recover $m \equiv c^d \equiv (53384)^{60343} \equiv 34152 \pmod{n}$.
- Decryption by an exponent d' other than d does not give back m. For example, take $d' = 38367$. We have $m' \equiv c^{d'} \equiv (53384)^{38367} \equiv 71303 \pmod{n}$.

Why RSA Works?

- Assume that $m \in \mathbb{Z}_n^*$. By Euler's theorem, $m^{\phi(n)} \equiv 1 \pmod{n}$.
- Now, $ed \equiv 1 \pmod{\phi(n)}$, that is, $ed = 1 + k\phi(n)$ for some integer k. Therefore,

$$c^d \equiv m^{ed} \equiv m^{1+k\phi(n)} \equiv m \times \left(m^{\phi(n)}\right)^k \equiv m \times 1^k \equiv m \pmod{n}.$$

- **Note:** The message can be recovered uniquely even when $m \notin \mathbb{Z}_n^*$.

Security of RSA

- If n can be factored, $\phi(n)$ can be computed and so d can be determined from e by extended gcd computation. Once d is known, any ciphertext can be decrypted.
- At present, no other method is known to decrypt RSA-encrypted messages.
- RSA derives security from the intractability of the IFP.
- If e, d, n are known, there exists a probabilistic polynomial-time algorithm to factor n. So RSA key inversion is as difficult as IFP. But RSA decryption without the knowledge of d *may be* easier than factoring n.
- In practice, we require the size of n to be $\geqslant 1024$ bits with each of p, q having nearly half the size of n.

How to Speed Up RSA?

Encryption: Take small encryption exponent e (like the smallest prime not dividing $\phi(n)$).

Decryption:

- Small decryption exponents invite many attacks.
- Store $n, e, d, p, q, d_1, d_2, h$, where $d_1 = d \operatorname{rem} (p-1)$, $d_2 = d \operatorname{rem} (q-1)$ and $h = q^{-1} \pmod{p}$.
- Carry out decryption as:
  - $m_1 = c^{d_1} \pmod{p}$.
  - $m_2 = c^{d_2} \pmod{q}$.
  - $t = h(m_1 - m_2) \pmod{p}$.
  - $m = m_2 + tq$.
- A speedup of about 4 is obtained.

ElGamal Encryption

- **Key generation**
  The recipient selects a random big prime p and a primitive root g modulo p, chooses a random $d \in \{2, 3, \ldots, p-2\}$, and computes $y \equiv g^d \pmod{p}$.
  Public key: (p, g, y).
  Private key: (p, g, d).
- **Encryption**
  Input: Plaintext $m \in \mathbb{Z}_p$ and recipient's public key (p, g, y).
  Output: Ciphertext (s, t).
  Generate a random integer $d' \in \{2, 3, \ldots, p-2\}$.
  Compute $s \equiv g^{d'} \pmod{p}$ and $t \equiv my^{d'} \pmod{p}$.
- **Decryption**
  Input: Ciphertext (s, t) and recipient's private key (p, g, d).
  Output: Recovered plaintext $m \equiv ts^{-d} \pmod{p}$.

ElGamal Encryption (contd)

- **Correctness:** We have $s \equiv g^{d'} \pmod{p}$ and $t \equiv my^{d'} \equiv m(g^d)^{d'} \equiv mg^{dd'} \pmod{p}$. Therefore, $m \equiv tg^{-dd'} \equiv t(g^{d'})^{-d} \equiv ts^{-d} \pmod{p}$.
- Example of ElGamal encryption
  - Take $p = 91573$ and $g = 67$. The recipient chooses $d = 23632$ and so $y \equiv (67)^{23632} \equiv 87955 \pmod{p}$.
  - Let $m = 29485$ be the message to be encrypted. The sender chooses $d' = 1783$ and computes $s \equiv g^{d'} \equiv 52958 \pmod{p}$ and $t \equiv my^{d'} \equiv 1597 \pmod{p}$.
  - The recipient retrieves $m \equiv ts^{-d} \equiv 1597 \times (52958)^{-23632} \equiv 29485 \pmod{p}$.

Security of ElGamal Encryption

- An eavesdropper knows g, p, y, s, t, where $y \equiv g^d \pmod{p}$ and $s \equiv g^{d'} \pmod{p}$. Determining m from (s, t) is equivalent to computing $g^{dd'} \pmod{p}$, since $t \equiv mg^{dd'} \pmod{p}$. (Here, m is masked by the quantity $g^{dd'} \pmod{p}$.) But d, d' are unknown to the attacker. So the ability to solve the DHP lets the eavesdropper break ElGamal encryption.
- Practically, we require p to be of size $\geqslant 1024$ bits for achieving a good level of security.

Probabilistic Encryption

To generate different ciphertext messages in different runs (for the same public key and plaintext message)

Goldwasser-Micali Encryption

- **Quadratic residuosity problem:** For a composite integer n and for an a with $\left(\frac{a}{n}\right) = 1$, determine whether a is a quadratic residue modulo n, that is, the whether the congruence $x^2 \equiv a \pmod{n}$ is solvable.
- Suppose $n = pq$ (product of two primes). $\left(\frac{a}{n}\right) = 1$ implies either $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1$ (a is a quadratic residue) or $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$ (a is a quadratic non-residue).
- We know no methods other than factoring n to solve this problem.

Goldwasser-Micali Encryption: Key Generation

- Choose two large primes p and q (of bit size $\geqslant 512$), and let $n = pq$.
- Generate random integers a, b with $\left(\frac{a}{p}\right) = \left(\frac{b}{q}\right) = -1$.
- Use CRT to generate $x \pmod{n}$ with $x \equiv a \pmod{p}$ and $x \equiv b \pmod{q}$.
- The Public key is (n, x), and the private key is p.

Goldwasser-Micali Encryption

Encryption

- The input is the r-bit plaintext message $m_1 m_2 \ldots m_r$.
- For each $i = 1, 2, \ldots, r$, choose $a_i \in \mathbb{Z}_n^*$ randomly and compute $c_i = x^{m_i} a_i^2 \pmod{n}$.
- The ciphertext message is the r-tuple $(c_1, c_2, \ldots, c_r) \in (\mathbb{Z}_n^*)^r$.

Decryption

- For $i = 1, 2, \ldots, r$, take

$$m_i = 0 \text{ if } \left(\frac{c_i}{p}\right) = 1, \text{ or}$$

$$m_i = 1 \text{ if } \left(\frac{c_i}{p}\right) = -1.$$

Goldwasser-Micali Encryption (contd)

Correctness

- If $m_i = 0$, then $c_i = a_i^2 \pmod{n}$ is a quadratic residue modulo n (and so modulo p and q also).
- If $m_i = -1$, then $c_i = xa_i^2 \pmod{n}$ Is a quadratic non-residue modulo n (or modulo p and q).

Remarks

- **Probabilistic encryption:** The ciphertext c_i depends on the choice of a_i.
- **Message expansion:** An r-bit plaintext message generates an rl-bit ciphertext message, where $l = |n|$.
- Without the knowledge of p (the private key), we do not know how to determine whether c_i is a quadratic residue or not modulo n.

Goldwasser-Micali Encryption: Example

Key generation

- Take $p = 653$ and $q = 751$, so $n = pq = 490403$.
- Take $a = 159$ and $b = 432$, so $x \equiv 313599 \pmod{n}$.
- The public-key is $(490403, 313599)$ and the private key is 653.

Encryption

- Let us encrypt the 3-bit message $m_1m_2m_3 = 101$.
- Choose $a_1 = 356217$ and compute $c_1 \equiv xa_1^2 \equiv 398732 \pmod{n}$.
- Choose $a_2 = 159819$ and compute $c_2 \equiv a_2^2 \equiv 453312 \pmod{n}$.
- Choose $a_3 = 482474$ and compute $c_3 \equiv xa_3^2 \equiv 12380 \pmod{n}$.

Decryption

- $\left(\frac{398732}{p}\right) = -1$, so $m_1 = 1$.
- $\left(\frac{453312}{p}\right) = 1$, so $m_2 = 0$.
- $\left(\frac{12380}{p}\right) = -1$, so $m_3 = 1$.

Diffie-Hellman Key Exchange

- Alice and Bob decide about a prime p and a primitive root g modulo p.
- Alice generates a random $a \in \{2, 3, \ldots, p-2\}$ and sends $g^a \pmod{p}$ to Bob.
- Bob generates a random $b \in \{2, 3, \ldots, p-2\}$ and sends $g^b \pmod{p}$ to Alice.
- Alice computes $g^{ab} \equiv (g^b)^a \pmod{p}$.
- Bob computes $g^{ab} \equiv (g^a)^b \pmod{p}$.
- The quantity $g^{ab} \pmod{p}$ is the secret shared by Alice and Bob.

- The Diffie-Hellman protocol works in other groups (finite extension fields and elliptic curve groups).

Diffie-Hellman Key Exchange: Example

- Alice and Bob first take $p = 91573$, $g = 67$.
- Alice generates $a = 39136$ and sends $g^a \equiv 48745 \pmod{p}$ to Bob.
- Bob generates $b = 8294$ and sends $g^b \equiv 69167 \pmod{p}$ to Alice.
- Alice computes $(69167)^{39136} \equiv 71989 \pmod{p}$.
- Bob computes $(48745)^{8294} \equiv 71989 \pmod{p}$.
- The secret shared by Alice and Bob is 71989.

Diffie-Hellman Key Exchange: Security

- An eavesdropper knows p, g, g^a, g^b and desires to compute $g^{ab} \pmod p$, that is, the eavesdropper has to solve the DHP.
- If discrete logs can be computed in $\mathbb{Z}_p^*$, then a can be computed from g^a and one subsequently obtains $g^{ab} \equiv (g^b)^a \pmod p$. So algorithms for solving the DLP can be used to break DH key exchange.
- Breaking DH key exchange *may be* easier than solving DLP.
- At present, no method other than computing discrete logs in $\mathbb{Z}_p^*$ is known to break DH key exchange.
- Practically, we require p to be of size $\geqslant 1024$ bits. The security does not depend on the choice of g. However, a and b must be sufficiently randomly chosen.

Common Signature Algorithms

Signature algorithm	Security depends on
RSA signature	Integer factoring problem
ElGamal encryption	DLP
Rabin signature	Square-root problem
Schnorr signature	DLP
Nyberg-Rueppel signature	DLP
Digital signature algorithm (DSA)	DLP
Elliptic curve version of DSA (ECDSA)	DLP in elliptic curves
XTR signature	DLP
NTRUSign	Closest vector problem

Digital Signatures: Classification

- **Deterministic signatures:** For a given message the same signature is generated on every occasion the signing algorithm is executed.
- **Probabilistic signatures:** On different runs of the signing algorithm different signatures are generated, even if the message remains the same.
- Probabilistic signatures offer better protection against some kinds of forgery.
- Deterministic signatures are of two types:
  - **Multiple-use signatures:** Slow. Parameters are used multiple times.
  - **One-time signatures:** Fast. Parameters are used only once.

RSA Signature

- **Key generation**
  The signer generates two random large primes p, q, computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, finds a random integer e with $\gcd(e, \phi(n)) = 1$, and determines an integer d with $ed \equiv 1 \pmod{\phi(n)}$.

  Public key: (n, e).
  Private key: (n, d).
- **Signature generation**
  Input: Message $m \in \mathbb{Z}_n$ and signer's private key (n, d).
  Output: Signed message (m, s) with $s \equiv m^d \pmod{n}$.
- **Signature verification**
  Input: Signed message (m, s) and signer's public key (n, e).
  Output: "Signature verified" if $s^e \equiv m \pmod{n}$,
  "Signature not verified" if $s^e \not\equiv m \pmod{n}$.

RSA Signature: Example

- Let $p = 257$, $q = 331$, so that $m = pq = 85067$ and $\phi(n) = (p-1)(q-1) = 84480$. Take $e = 19823$, so that $d \equiv e^{-1} \equiv 71567 \pmod{\phi(n)}$.

  Public key: $(85067, 19823)$.

  Private key: $(85067, 71567)$.
- Let $m = 3759$ be the message to be signed. Generate $s \equiv m^d \equiv 13728 \pmod{n}$. The signed message is $(3759, 13728)$.
- Verification of $(m, s) = (3759, 13728)$ involves the computation of $s^e \equiv (13728)^{19823} \equiv 3759 \pmod{n}$. Since this equals m, the signature is verified.
- Verification of a forged signature $(m, s) = (3759, 42954)$ gives $s^e \equiv (42954)^{19823} \equiv 22968 \pmod{n}$. Since $s^e \not\equiv m \pmod{n}$, the forged signature is not verified.

ElGamal Signature

- **Key generation**
  Like ElGamal encryption, one chooses p, g and computes a key-pair (y, d) where $y \equiv g^d \pmod{p}$. The public key is (p, g, y), and the private key is (p, g, d).
- **Signature generation**
  Input: Message $m \in \mathbb{Z}_p$ and signer's private key (p, g, d).
  Output: Signed message (m, s, t).
  Generate a random session key $d' \in \{2, 3, \ldots, p-2\}$.
  Compute $s \equiv g^{d'} \pmod{p}$ and
  $t \equiv d'^{-1}(H(m) - dH(s)) \pmod{p-1}$.
- **Signature verification**
  Input: Signed message (m, s, t) and signer's public key (p, g, y).
  Set $a_1 \equiv g^{H(m)} \pmod{p}$ and $a_2 \equiv y^{H(s)} s^t \pmod{p}$.
  Output "signature verified" if and only if $a_1 = a_2$.

ElGamal Signature (contd)

- **Correctness:** $H(m) \equiv dH(s) + td' \pmod{p-1}$. So $a_1 \equiv g^{H(m)} \equiv (g^d)^{H(s)}(g^{d'})^t \equiv y^{H(s)}s^t \equiv a_2 \pmod{p}$.
- **Example:**
    - Take $p = 104729$ and $g = 89$. The signer chooses the private exponent $d = 72135$ and so $y \equiv g^d \equiv 98771 \pmod{p}$.
    - Let $m = 23456$ be the message to be signed. The signer chooses the session exponent $d' = 3951$ and computes $s \equiv g^{d'} \equiv 14413 \pmod{p}$ and $t \equiv d'^{-1}(m - ds) \equiv (3951)^{-1}(23456 - 72135 \times 14413) \equiv 17515 \pmod{p-1}$.
    - Verification involves computation of $a_1 \equiv g^m \equiv 29201 \pmod{p}$ and $a_2 \equiv y^s s^t \equiv (98771)^{14413} \times (14413)^{17515} \equiv 29201 \pmod{p}$. Since $a_1 = a_2$, the signature is verified.

ElGamal Signature (contd)

- **Forging:** A forger chooses $d' = 3951$ and computes $s \equiv g^{d'} \equiv 14413 \pmod p$. But computation of t involves d which is unknown to the forger. So the forger randomly selects $t = 81529$. Verification of this forged signature gives $a_1 \equiv g^m \equiv 29201 \pmod p$ as above. But $a_2 \equiv y^s s^t \equiv (98771)^{14413} \times (14413)^{81529} \equiv 85885 \pmod p$, that is, $a_1 \neq a_2$ and the forged signature is not verified.
- **Security:**
  - Computation of s can be done by anybody. However, computation of t involves the signer's private exponent d. If the forger can solve the DLP modulo p, then d can be computed from the public-key y, and the correct signature can be generated.
  - The prime p should be large (of bit-size $\geqslant 1024$) in order to preclude this attack.

Digital Signature Algorithm (DSA)

Accepted by the US Government as a standard.

Parameter generation

- Generate a prime p of bit length $512 + 64\lambda$ for $0 \leqslant \lambda \leqslant 8$.
- $p - 1$ must have a prime divisor r of bit length 160.
- A specific algorithm is recommended for computing p and r.
- Compute an element $g \in \mathbb{F}_p^*$ with multiplicative order r.
- Make p, r, g public.

Key generation

- Generate a random $d \in \{2, 3, \ldots, r - 1\}$ (private key).
- Compute $y \equiv g^d \pmod{p}$ (public key).

DSA (contd)

Signature generation

- To sign a message M, proceed as follows:
- Generate random session key $d' \in \{2, 3, \ldots, r-1\}$.
- Compute $s = \left(g^{d'} \pmod{p}\right) \pmod{r}$ and
  $t = d'^{-1}(H(M) + ds) \pmod{r}$.
- Output the signed message (M, s, t).

Signature verification

- To verify a signature (M, s, t) using the signer's public key y:
- If s or t is not in $\{0, 1, \ldots, r-1\}$, return "not verified".
- Compute $w \equiv t^{-1} \pmod{r}$, $w_1 \equiv H(M)w \pmod{r}$, and
  $w_2 \equiv sw \pmod{r}$.
- Compute $\tilde{s} = (g^{w_1} y^{w_2} \pmod{p}) \pmod{r}$.
- Signature is verified if and only if $\tilde{s} = s$.

DSA (contd)

Correctness

- $w \equiv t^{-1} \equiv d'(H(M) + ds)^{-1} \pmod{r}$.
- $g^{w_1} y^{w_2} \equiv g^{w_1 + dw_2} \equiv g^{(H(M)+ds)w} \equiv g^{d'} \pmod{p}$.
- Consequently, $\tilde{s} \equiv s \pmod{r}$.

Remarks

- Although the modulus p may be as long as 1024 bits, the signature size (s, t) is only 320 bits.
- The security of DSA depends on the difficulty of solving the DLP in $\mathbb{F}_p^*$.

DSA: Example

Parameters:

$p = 21101$, $r = 211$, and $g \equiv 12345^{(p-1)/r} \equiv 17808 \pmod{p}$.

Key pair:

$d = 79$ [private key] and $y \equiv g^d \equiv 2377 \pmod{p}$ [public key].

Signature generation:

- To sign $M = 8642$.
- Choose $d' = 167$.
- Compute $s = \left(g^{d'} \pmod{p}\right) \pmod{r} = 13687 \operatorname{rem} r = 183$.
- Compute $t \equiv d'^{-1}(M + ds) \equiv 132 \pmod{r}$.
- The signature is the pair $(183, 132)$.

DSA: Example (contd)

Signature verification:

- To verify $(8642, 183, 132)$.
- Compute $w \equiv t^{-1} \equiv 8 \pmod{r}$.
- Compute $w_1 \equiv Mw \equiv 8642 \times 8 \equiv 139 \pmod{r}$.
- Compute $w_2 \equiv sw \equiv 183 \times 8 \equiv 198 \pmod{r}$.
- $g^{w_1} y^{w_2} \equiv 17808^{139} \times 2377^{198} \equiv 13687 \pmod{p}$.
- $\tilde{s} = 13687 \operatorname{rem} r = 183 = s$, so signature is verified.

Verification of faulty signature:

- To verify $(8642, 138, 123)$.
- Compute $w \equiv t^{-1} \equiv 199 \pmod{r}$.
- Compute $w_1 \equiv Mw \equiv 8642 \times 199 \equiv 108 \pmod{r}$.
- Compute $w_2 \equiv sw \equiv 138 \times 199 \equiv 32 \pmod{r}$.
- $g^{w_1} y^{w_2} \equiv 17808^{108} \times 2377^{32} \equiv 3838 \pmod{p}$.
- $\tilde{s} = 3838 \operatorname{rem} r = 40 \neq s$, so signature is not verified.

Elliptic Curve Digital Signature Algorithm (ECDSA)

Parameter generation

Input: A finite field $\mathbb{F}_q$ with $q = p \in \mathbb{P}$ or $q = 2^m$.

1. Choose $a, b \in \mathbb{F}_q$ randomly.
2. Let $E : \begin{cases} y^2 = x^3 + ax + b & \text{if } q = p, \\ y^2 + xy = x^3 + ax^2 + b & \text{if } q = 2^m. \end{cases}$
3. Compute the size n of $E(\mathbb{F}_q)$.
4. If n has no prime divisor $r > \max(2^{160}, 4\sqrt{q})$, go to Step 1.
5. If $n \mid (q^k - 1)$ for $k \in \{1, 2, \ldots, 20\}$ (MOV attack), go to Step 1.
6. If $n = q$ (anomalous attack), go to Step 1.
7. Choose $P' \in E(\mathbb{F}_q)$ randomly.
8. Compute $P = (n/r)P'$.
9. If $P = \mathcal{O}$, go to Step 7.
10. Return E, n, r, P.

ECDSA (contd)

ECDSA keys:

- Choose $d \in \{2, 3, \ldots, r-1\}$ randomly [private key].
- Compute $Y = dP \in E(\mathbb{F}_q)$ [public key].

Signature generation: (To sign message M)

- Choose session key $d' \in \{2, 3, \ldots, r-1\}$.
- Compute $(h, k) = d'P \in E(\mathbb{F}_q)$.
- Take $s = h \pmod r$ and $t = d'^{-1}(H(M) + ds) \pmod r$.
- Output the signed message (M, s, t).

Signature verification: (To verify signed message (M, s, t))

- If s or t is not in $\{1, 2, \ldots, r-1\}$, return "not verified".
- Compute $w \equiv t^{-1} \pmod r$, $w_1 \equiv H(M)w \pmod r$ and $w_2 \equiv sw \pmod r$.
- Compute the point $Q = w_1P + w_2Y \in E(\mathbb{F}_q)$.
- If $Q = \mathcal{O}$, return "not verified".
- Let $Q = (\tilde{h}, \tilde{k})$. Compute $\tilde{s} = \tilde{h} \pmod r$.
- Signature is verified if and only if $\tilde{s} = s$.

Blind Signatures

A signer Bob signs a message m without knowing m.
Blind signatures insure anonymity during electronic payment.

Chaum's blind RSA signature

Input: A message M generated by Alice.
Output: Bob's blind RSA signature on M.
Steps:

- Alice gets Bob's public-key (n, e).
- Alice computes $m = H(M) \in \mathbb{Z}_n$.
- Alice sends to Bob the masked message $m' \equiv \rho^e m \pmod{n}$ for a random ρ.
- Bob sends the signature $\sigma = m'^d \pmod{n}$ back to Alice.
- Alice computes Bob's signature $s \equiv \rho^{-1}\sigma \pmod{n}$ on M.

Correctness of Chaum's Blind RSA Signature

- Assume that $\rho \in \mathbb{Z}_n^*$.
- Since $ed \equiv 1 \pmod{\phi(n)}$, we have $\sigma \equiv m'^d \equiv (\rho^e m)^d \equiv \rho^{ed} m^d \equiv \rho m^d \pmod{n}$.
- Therefore, $s \equiv \rho^{-1}\sigma \equiv m^d \equiv H(M)^d \pmod{n}$.

Undeniable Signatures

- Active participation of the signer is necessary during verification.
- A signer is not allowed to deny a legitimate signature made by him.
- An undeniable signature comes with a **denial** or **disavowal protocol** that generates one of the following three outputs:
  - Signature verified
  - Signature forged
  - The signer is trying to deny his signature by not participating in the protocol properly.

Examples

- Chaum-van Antwerpen undeniable signature scheme
- RSA-based undeniable signature scheme

Entity Authentication

Weak Authentication: Passwords

Set-up phase

- Alice supplies a secret password P to Bob.
- Bob transforms (typically encrypts) P to generate $Q = f(P)$.
- Bob stores Q for future use.

Authentication phase

- Alice supplies her password P' to Bob.
- Bob computes $Q' = f(P')$.
- Bob compares Q' with the stored value Q.
- $Q' = Q$ if and only if $P' = P$.
- If $Q' = Q$, Bob accepts Alice's identity.

Passwords (contd)

- It should be difficult to invert the initial transform $Q = f(P)$.
- Knowledge of Q, even if readable by enemies, does not reveal P.

Drawbacks

- Alice reveals P itself to Bob. Bob may misuse this information.
- P resides in unencrypted form in the memory during the authentication phase. A third party having access to this memory obtains Alice's secret.

Challenge-Response Authentication

- Also known as **strong authentication**.
- Possession of a secret by a claimant is proved to a verifier.
- The secret is not revealed to the verifier.
- One of the parties sends a challenge to the other.
- The other responds to the challenge appropriately.
- This conversation does not reveal any information about the secret to the verifier or to an eavesdropper.

Challenge-Response Scheme Using Encryption

The protocol

- Alice wants to prove to Bob her knowledge of the private key d in the key-pair (e, d).
- Bob generates a random bit string r and computes $w = H(r)$.
- Bob reads Alice's public key e and computes $c = f_e(r, e)$.
- Bob sends the **challenge** (w, c) to Alice.
- Alice computes $r' = f_d(c, d)$.
- If $H(r') \neq w$, Alice quits the protocol.
- Alice sends the **response** r' to Bob.
- Bob accepts Alice's identity if and only if $r' = r$.

Correctness

- Bob checks whether Alice can correctly decrypt the challenge c.
- Bob sends w as a **witness** of his knowledge of r.
- Before sending the decrypted plaintext r', Alice confirms that Bob actually knows the plaintext r.

Challenge-Response Scheme Using Signature

The protocol

- Alice wants to prove to Bob her knowledge of the private key d in the key-pair (e, d).
- Bob sends a random string r_B to Alice.
- Alice generates a random string r_A and signs $s = f_d(r_A \,||\, r_B, d)$.
- Alice sends (r_A, s) to Bob.
- Bob generates $r'_A \,||\, r'_B = f_e(s, e)$ using Alice's public key e.
- Bob accepts Alice's identity if and only if $r'_A = r_A$ and $r'_B = r_B$.

Correctness

- The signature s can be generated only by a party who knows d.
- Use of r_B prevents **replay attacks** by an eavesdropper.
- Use of **timestamps** achieves the same objective. Alice signs $s = f_d(t_A, d)$ and sends (t_A, s) to Bob. Bob retrieves $t'_A = f_e(s, e)$. Bob accepts Alice if and only if $t_A = t'_A$ and t_A is a valid timestamp.

Zero-Knowledge Protocols (ZKP)

- A ZKP is a strong authentication scheme with a mathematical proof that no information is leaked to the verifier or a listener during an authentication interaction.
- Alice (the claimant) chooses a random **commitment** and sends a **witness** of the commitment to Bob (the verifier).
- Bob sends a random **challenge** to Alice.
- Alice sends a **response** to the challenge, back to Bob.
- If Alice knows the secret, she can succeed in the protocol.
- A listener can succeed with a probability $P \ll 1$.
- The protocol may be repeated multiple times (t times), so that the probability of success for an eavesdropper (P^t) can be made as small as desirable.

Feige-Fiat-Shamir (FFS) Protocol

Selection of Alice's secrets

- The following steps are performed by Alice or a trusted third party.
- Select two large distinct primes p and q each congruent to 3 modulo 4.
- Compute $n = pq$, and select a small integer $t = O(\ln \ln n)$.
- Make n and t public.
- Select t random integers $x_1, x_2, \ldots, x_t \in \mathbb{Z}_n^*$.
- Select t random bits $\delta_1, \delta_2, \ldots, \delta_t \in \{0, 1\}$.
- Compute $y_i \equiv (-1)^{\delta_i} (x_i^2)^{-1} \pmod{n}$ for $i = 1, 2, \ldots, t$.
- Make $y_1, y_2, \ldots, y_t$ public. Keep $x_1, x_2, \ldots, x_t$ secret.

Feige-Fiat-Shamir Protocol: Authentication

Alice wants to prove to Bob her knowledge of the secrets $x_1, x_2, \ldots, x_t$.

- **[Commitment]** Alice selects random $c \in \mathbb{Z}_n^*$ and $\gamma \in \{0, 1\}$.
- **[Witness]** Alice sends $w \equiv (-1)^\gamma c^2 \pmod{n}$ to Bob.
- **[Challenge]** Bob sends random bits $\epsilon_1, \epsilon_2, \ldots, \epsilon_t$ to Alice.
- **[Response]** Alice sends $r \equiv c \prod_{i=1}^{t} x_i^{\epsilon_i} \pmod{n}$ to Bob.
- **[Authentication]** Bob computes $w' \equiv r^2 \prod_{i=1}^{t} y_i^{\epsilon_i} \pmod{n}$, and accepts Alice's identity if and only if $w' \neq 0$ and $w' \equiv \pm w \pmod{n}$.

Feige-Fiat-Shamir Protocol: Correctness

- Since $r \equiv c \prod_{i=1}^{t} x_i^{\epsilon_i} \pmod{n}$, we have $r^2 \equiv c^2 \prod_{i=1}^{t} (x_i^2)^{\epsilon_i} \pmod{n}$.
- But $y_i \equiv (-1)^{\delta_i} (x_i^2)^{-1} \pmod{n}$.
- Therefore, $w' \equiv r^2 \prod_{i=1}^{t} y_i^{\epsilon_i} \equiv c^2 \prod_{i=1}^{t} (-1)^{\epsilon_i \delta_i} \pmod{n}$, whereas $w \equiv (-1)^{\gamma} c^2 \pmod{n}$.
- Consequently, $w' \equiv \pm w \pmod{n}$.
- The check $w' \neq 0$ eliminates the commitment $c = 0$ which succeeds always irrespective of the knowledge of the secrets x_i.

Feige-Fiat-Shamir Protocol: Security

- Consider the simple case $t = 1$.
- Alice sends c or cx_1 as the response r, depending on whether $\epsilon = 0$ or $\epsilon = 1$. Ability to send both is equivalent to knowing x_1.
- Computing c from w requires computing square roots modulo the composite n with unknown factors.
- In an attempt to impersonate Alice, an eavesdropper Carol may choose any random c and send the witness $w \equiv (-1)^{\gamma}c^2 \pmod{n}$.
- If Bob chooses $\epsilon = 0$, Carol can send the correct response c. But if Bob chooses $\epsilon = 1$, Carol needs to know x_1 to send the correct response cx_1.
- Carol may succeed in sending the correct response c to the challenge $\epsilon_1 = 1$ by arranging the witness improperly as $w \equiv (-1)^{\gamma}c^2y_1^{-1} \pmod{n}$. But the challenge $\epsilon_1 = 0$ now requires the knowledge of x_1 to compute the correct response $cx_1^{-1} \pmod{n}$ corresponding to the improper witness.
- In either case, the success probability is nearly $1/2$.

Guillou-Quisquater (GQ) Protocol

- Alice generates an RSA-based exponent-pair (e, d) under the modulus n.
- Alice chooses a random $m \in \mathbb{Z}_n^*$ and computes $s \equiv m^{-d} \pmod{n}$. Alice makes m public and keeps s secret. Alice tries to prove to Bob her knowledge of s.
- **The protocol**

Alice selects a random $c \in \mathbb{Z}_n^*$.	[Commitment]
Alice sends to Bob $w \equiv c^e \pmod{n}$.	[Witness]
Bob sends to Alice a random $\epsilon \in \{1, 2, \ldots, e\}$.	[Challenge]
Alice sends to Bob $r \equiv cs^\epsilon \pmod{n}$.	[Response]

Bob computes $w' \equiv m^\epsilon r^e \pmod{n}$.
Bob accepts Alice's identity if and only if $w' \neq 0$ and $w' = w$.

Guillou-Quisquater Protocol (contd)

- **Correctness**

  $w' \equiv m^{\epsilon} r^{e} \equiv m^{\epsilon}(cs^{\epsilon})^{e} \equiv m^{\epsilon}(cm^{-d\epsilon})^{e} \equiv (m^{1-ed})^{\epsilon} c^{e} \equiv c^{e} \equiv w \pmod{n}$.

- **Security**
  - The quantity s^{ϵ} is blinded by the random commitment c.
  - As a witness for c, Alice presents its encrypted version w.
  - Bob (or an eavesdropper) cannot decrypt w to compute c and subsequently s^{ϵ}.
  - An eavesdropper's guess about ϵ is successful with probability $1/e$.
  - The check $w' \neq 0$ precludes the case $c = 0$ which lets a claimant succeed always.

Digital Certificates: Introduction

- Bind public-keys to entities.
- Required to establish the authenticity of public keys.
- Guard against malicious public keys.
- Promote confidence in using others' public keys.
- Require a **Certification Authority** (CA) whom every entity over a network can believe. Typically, a government organization or a reputed company can be a CA.
- In case a certificate is compromised, one requires to revoke it.
- A revoked certificate cannot be used to establish the authenticity of a public key.

Digital Certificates: Contents

- A digital certificate contains particulars about the entity whose public key is to be embedded in the certificate:
  - Name, address and other personal details of the entity.
  - The public key of the entity. The key pair may be generated by either the entity or the CA. If the CA generates the key pair, then the private key is handed over to the entity by trusted couriers.

  The certificate is digitally signed by the private key of the CA.
- If signatures cannot be forged, nobody other than the CA can generate a valid certificate for an entity.

Digital Certificates: Revocation

- A certificate may become invalid due to several reasons:
  - Expiry of the certificate
  - Possible or suspected compromise of the entity's private key
  - Detection of malicious activities of the owner of the certificate
- An invalid certificate is revoked by the CA.
- **Certificate Revocation List** (**CRL**): The CA maintains a list of revoked certificates.
- If Alice wants to use Bob's public key, she obtains the certificate for Bob's public key. If the CA's signature is verified on this certificate and if the certificate is not found in the CRL, then Alice gains the desired confidence to use Bob's public key.