CRYPTOGRAPHY LECTURE, 2023,

Master level

Mirosław Kutyłowski

Asymmetric Encryption

Fundamental difference

- decryption and encryption key different
- one way relation: $\text{encryption\_key} := F(\text{decryption\_key})$
  - decryption key: private, usual notation: sk
  - encryption key: public (≈not secret), usual notation: pk

$$\text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(M)) = M$$

Other options:

multiple decryption keys:

→ in order to recover the plaintext all decryption keys must be used (multiparty protocol)

- splitting risk of key capture: two or more devices involved
- example: e-voting (cascade of decryption servers):

$$\mathrm{Dec}_{\mathrm{sk}_2}(\mathrm{Dec}_{\mathrm{sk}_1}(\mathrm{Enc}_{\mathrm{pk}}(M)) = M$$

Asymmetric encryption versus CPA-IND

- everybody can encrypt, so automatically in the scenario of attack
- after presenting $C = \mathrm{Enc}_{\mathrm{pk}}(M_b)$ and M_0, M_1 the adversary could encrypt M_0, M_1 and compare with C – winning the game for deterministic Enc

So Enc should be non-deterministic, with high entropy

CCA-IND

- decryption scenario such as for symmetric encryption (requires secret key)
- deriving sk from pk would immediately let the adversary to win CCA-IND and CPA-IND games

ElGamal public key encryption

Keys: sk generated at random, $\mathrm{pk} := g^x$

Encryption of m

i. choose k at random

ii. $C := (\mathrm{pk}^k \cdot m,\, g^k)$

Decryption of $C = (a, b)$

calculate $m := a/b^{\mathrm{sk}}$

correctness: $\frac{a}{b^{\mathrm{sk}}} = \frac{\mathrm{pk}^k \cdot m}{(g^k)^{\mathrm{sk}}} = \frac{\mathrm{pk}^k \cdot m}{(g^{\mathrm{sk}})^k} = \frac{\mathrm{pk}^k \cdot m}{\mathrm{pk}^k} = m$

Which assumption needed?

Security of ElGamal

one can decrypt $\Rightarrow$ one can solve CDH:

given (g, g^a, g^k)

- create a "ciphertext" (z, g^k) for z chosen at random
- use algorithm deriving the corresponding plaintext m for public key g^a
- then $z = g^{a \cdot k} \cdot m$, so $g^{a \cdot k} = z/m$

Remark: there are groups where DDH is easy but CDH is hard

DDH easy $\Rightarrow$ ElGamal encryption broken

– given a ciphertext (a, b) and a candidate plaintext m

– **goal:** check if (a, b) encrypts m

– take $(g, \mathrm{pk}, b, a/m)$ and test via DDH Oracle: "yes"$\Leftrightarrow$ this is a ciphertext of m

Security of ElGamal versus KEA1 Assumption

– given a ciphertext (a, b) and the public key pk

– successful decryption is equivalent to computing a tuple $(g, \mathrm{pk}, b, a/m)$

$$(g, \mathrm{pk}, b, a/m) = (g, \mathrm{pk}, g^k, \mathrm{pk}^k)$$

KEA1

given (a, b, c) is it possible to create (a, b, c, d) which is a DH tuple?

KEA1

in some cases it is possible:

- if you know k such that $b = a^k$ then it suffices to take $d := c^k$
- are there any other possibilities if c is chosen at random?

KEA1 Assumption:

if for a, b and c – where c is chosen at random — you can provide d such that (a, b, c, d) is a DH tuple

$\Rightarrow$

you may run an Extractor that yields k such that $b = a^k$

Remarks:

- "no situations in gray zone":, either you know an exponent and can create DH tuple or you cannot
- be careful: similar assumptions turn out to be false
- KEA1 is in practice the basic assumption for many schemes

ElGamal re-encryption

given a ciphertext (a, b) for public key pk one can re-encrypt it with a random t:

$$(c, d) := (a \cdot \mathrm{pk}^t, b \cdot g^t)$$

universal re-encryption: What if $\mathbf{pk}$ unknown?

– ciphertext of m

$$(\mathrm{pk}^k \cdot m, g^k, \mathrm{pk}^n, g^n)$$

– re-encryption of (a, b, c, d)

$$(a', b', c', d') := (a \cdot c^t, b \cdot d^t, c^u, d^u)$$

Mixing Server

input: ciphertexts $C_1, C_2, ..., C_m$

output: the same ciphertexts after re-encryption in a random order

Applications:

i. e-voting

ii. anonymous communication

$\Leftarrow$ cascade of re-encryption servers

Correctness of cascade of re-encryption servers:

Randomized Partial Checking

given cascade of MIX servers: $S_1, S_2, ..., S_m$ processing n ciphertexts

Phase 1

- the controller chooses $A \subset \{1, ..., n\}$ of cardinality $n/2$
- for each $i \in A$, server S_1 reveals re-encryption exponent for the ith ciphertext

  $\Rightarrow$ links to $n/2$ inputs of S_2 revealed: the controller re-encrypts and checks the result

Phase 2

- S_2 reveals the re-encryption exponents for those input ciphertexts that are not linked after phase 1

On a picture:

Result after RPC:

separate mixing

- the ciphertexts with index $\in A$
- the ciphertext with index $\notin A$

Then do the same for $S_2, S_3, S_4,$ then for S_4, S_5, S_6, ...

Identity based encryption

background: learning the public key of the recipient may require effort and Public Key Infrastructure

idea: user ID as the public key

how to make it real???

Pairings - algebraic tools

- groups G_1, G_2 and G_T , cyclic, generators g_1 and g_2 of G_1, G_2
- bilinear pairing mapping $e\colon G_1 \times G_2 \longrightarrow G_T$
  - **bilinearity:** $e(k \cdot A,\ m \cdot B) = e(A,\ B)^{k \cdot m}$ (additive notation in $G_1,\ G_2$ and multiplicative for G_T
  - **non-degenerate**: $e(g_1, g_2) \neq 1$ (in G_T)

  and e easy to compute

Classification: $G_1 = G_2$ – type 1 pairing

$G_1 \neq G_2$ but we know a homomorphism $h\colon G_1 \rightarrow G_2$ – type 2 pairing

no homomorphism between G_1 and G_2 is known - type 3 pairing

DDH and pairings

- DDH assumption is false for type-1 pairings:

  (A, B, C, D) is a DH tuple iff $e(B, C) = e(A, D)$

  → indeed, if $B = m \cdot A, \quad C = k \cdot A, \quad D = (k \cdot m) \cdot A$, then

$$e(B, C) = e(m \cdot A, k \cdot A) = e(A, A)^{m \cdot k}$$

$$e(A, D) = e(A, (k \cdot m) \cdot A) = e(A, A)^{m \cdot k}$$

nevertheless, CDH might be hard in G_1!

Identity based encryption (IBE) - example: Boneh-Franklin scheme

Key Generation Center – a user obtains a private key after authenticating themself against KGC

setup:

- pairing $e\colon G \times G \to G_T$, P - a generator of G
- master private key s for KGC, master public key: $K := s \cdot P$
- s random, K – public system parameter
- hash functions H_1 mapping into $G \setminus \{0\}$ and H_2 mapping from G_T

Generation of secret keys for the user:

user with **official identifier** ID:

(e.g. Personal Identity Number, registry number for enterprises...)

– user public key: $Q_{\mathrm{ID}} := H_1(\mathrm{ID})$ (element of group G)

– user secret key: $D_{\mathrm{ID}} := s \cdot Q_{\mathrm{ID}}$

(KGC must be honest!)

Encryption of message m for user ID

1. $Q_{\mathrm{ID}} := H_1(\mathrm{ID}),\ \ g_{\mathrm{ID}} := e(Q_{\mathrm{ID}}, K)$
2. choose r at random, $U := r \cdot P$
3. $v := m \oplus H_2(g_{\mathrm{ID}}^r)$
4. output (U, v)

Decryption of (U, v)

1. $z := e(U, D_{\mathrm{ID}})$ note that:

$$e(U, D_{\mathrm{ID}}) = e(r \cdot P, s \cdot Q_{\mathrm{ID}}) = e(P, Q_{\mathrm{ID}})^{r \cdot s} = e(s \cdot P, Q_{\mathrm{ID}})^{r} = e(K, Q_{\mathrm{ID}})^{r} = g_{\mathrm{ID}}^{r}$$

2. $m := v \oplus H_2(z)$

Security - Bilinear Diffie-Hellman Assumption (BDH)

given: $a \cdot P$, $b \cdot P$, $c \cdot P$

sought: $e(P, P)^{a \cdot b \cdot c}$

BDH Assumption

it is infeasible to solve BDH in a given group

Theorem

Boneh-Franklin IBE scheme is semantically secure for ROM provided that BDH Assumption holds.

RSA

- based on RSA numbers: $n = p \cdot q$, where p and q are large prime numbers
- the function $F(p, q) = p \cdot q$ is a one-way function for large primes p, q

Group used:

G - the elements co-prime with n with multiplication modulo n

- $\phi(n) = (p-1) \cdot (q-1)$ elements in G $\quad$ ($n - p - q + 1$)

computations possible according to Chinese Remainder Theorem:

$$a \quad \rightarrow \quad (a \bmod p, a \bmod q)$$

computing $z = a \cdot b \bmod n$:

i. $z_p := a \cdot b \bmod p$

ii. $z_q := a \cdot b \bmod q$

iii. reconstruct z from z_p and z_q according to ChRT:

- compute m_p, m_q such that $m_p \cdot p + m_q \cdot q = 1$ according to Euclidean algorithm for GCD

- $z := z_p \cdot m_q \cdot q + z_q \cdot m_p \cdot p \bmod n$

RSA generation

i. choose odd number p of bitlength ... (at least 1024) at random

   1. test if p is prime (probabilistic prime number test)
   2. if not prime, then $p := p + 2$ and goto 1

ii. the same for q

iii. $n := p \cdot q$

Critical points:

- choice of initial values for the search : if predictable then p and q predictable
- consequence: something like 6% of RSA moduli in appear in more than 1 certificate of different owners
- failures of PRIME testing possible: especially if testing time reduced

Primality testing:

– step 1: fast sieve: test small factors for quick reject (most composite numbers have small factors!)

– step 2: probabilistic test

  example: **Miller-Rabin** test:

  background:

  - if n is prime, then $\mathbb{Z}_n^*$ is cyclic with $n-1$ elements, there are two roots of one: 1 and -1
  - if n is composite, then there at least 4 roots of 1

Algorithm of Miller-Rabin test

- repeat ... times:

  i. choose $a < n$ at random

  ii. $a := a^d \bmod n$

  iii. repeat until $a = -1 \bmod n$:

    - $a := a^2 \bmod n$
    - if $a = 1$ then return(composite) and abort

- return(prime)

where $n - 1 = 2^t \cdot d$

Issues

- this is a Monte Carlo algorithm: the output “prime” **can be incorrect**
- a single iteration witnesses that a composite number is composite with pbb $\frac{3}{4}$ or higher, but $\frac{3}{4}$ is the only guarantee
- to get a strong evidence many iterations needed

  moreover: operations on big numbers, many false candidates rejected until one n passes the test

$\Rightarrow$ many software products neglect the test and, for example, run only Fermat test:

choose a at random and test whether $a^{n-1} = 1 \bmod n$

(Fermat theorem holds for prime numbers n, ... but also for some composite numbers)

Encryption of m

1. $m_0 := \mathrm{encode}(m)$ - get a number $m_0 < n$ (from binary representation via some padding)
2. $\mathrm{Enc}_{n,e}(m) = m_0^e \bmod n$

Decryption of c

1. compute $m_0 := c^d \bmod n$
2. $m := \mathrm{encode}^{-1}(m_0)$

Magic

$$c^d = (m_0^e)^d = m_0^{e \cdot d} = m_0^{1+i\cdot(p-1)(q-1)} = m_0 \cdot m_0^{i(p-1)(q-1)} = m_0$$

the last equality follows from the fact that

– Z_n^* has $(p-1)(q-1)$ elements

– if a group has k elements, then $a^k = 1$ for each element a from the group (Euler's Theorem)

Manipulations

given a ciphertext c one can manipulate the plaintext

example: multiply the plaintext by 2:

1. compute $z := 2^e \bmod n$
2. calculate $c' := z \cdot c \bmod n$

the plaintext for c':

$$c'^d = (2^e \cdot c)^d = 2^{e \cdot d} \cdot c^d = 2 \cdot c^d = 2 \cdot \text{plaintext} \bmod n$$

OEAP-RSA encoding for RSA number n of bitlength N:

given: parameters k_0, k_1 , message m of length $N - k_0 - k_1$, hash functions G, F:

encoding procedure:

i. $m' =$ messsage m with k_1 zeroes appended: $m' = m00....0$

ii. generate k_0 bit string r at random

iii. $z := G(r)$ (output has $N - k_0$ bits)

iv. $X := m' \oplus z$

v. $Y := H(X) \oplus r$

vi. return X, Y

decoding:

i. $r := Y \oplus H(X)$

ii. $m' := X \oplus G(r)$ (if m' has no suffix of k_1 zeroes then reject, otherwise truncate zeroes)

Features of OEAP

1. for a random X, Y the decoding will abort with pbb $\approx 1/2^{k_1}$

   $k_1 = 40$ practically reduces CCA to CPA (the decryption oracle will return "error" repeatedly)

2. possibility for subliminal channel:

   parameter r can be chosen freely, for example:

$$r := \mathrm{Enc}_K(\text{hidden message})$$

RSA security

- not true that there is only one matching secret key:

  d and $d + \mathrm{LCM}(p-1, q-1)$ are equivalent

- factorization of $n \Rightarrow$ breaking public key

- finding private key gives factorization $\quad e \cdot d = 1 \bmod (p-1)(q-1)$)

$$e \cdot d = 1 + i \cdot (p-1)(q-1) = 1 + i \cdot (n - p - q + 1)$$

  i can be calculated, then we have $p + q$

$n = p \cdot ((\ldots - p))$ – equality of degree 2

But maybe it is possible to compute the plaintext without the secret key?

equivalent problem:

calculate the eth root of c is

RSA Assumption

it is infeasible unless you know d such that $e \cdot d = 1 \bmod (p-1)(q-1)$)

Post-quantum – example: McEllice

based on linear algebra, random error correcting codes

(n, k) – linear Error Correcting Codes:

- $n \times k$ generator matrix G given a word w of length k, its code is $G \cdot w^T$ of length n
- property needed: for every v the Hamming weight of $G \cdot v^T$ is either 0 or greater than t
- $\Rightarrow$ the minimal distance between codewords is at least $t + 1$:

$$G \cdot v^T \oplus G \cdot w^T = G \cdot (v \oplus w)^T$$

- decoding algorithms: different depending on the ECC

McEliece Encryption - key generation

1. choose a generator matrix G on $(n, k)-$ linear code (from some family) for correcting t errors
2. choose at random $k \times k$ non-singular matrix S
3. choose at random $n \times n$ permutation matrix P
4. $H := S \cdot G \cdot P$

public key: (H, t)

private key: S, P and decoding algorithm A corresponding to G

Encryption of m (k-bit string)

1. $c_0 := m \cdot H$
2. flip t bits of c_0 at random positions (creating t errors in the final code)

   $c := c_0 \oplus e$ where e is an error vector of Hamming weight t

Decryption

1. $c' := c \cdot P^{-1}$
2. decode the codeword c' with algorithm A to m'
3. $m := m' \cdot S^{-1}$

why the result is correct?

$$c \cdot P^{-1} = (c_0 \oplus e) \cdot P^{-1} = c_0 \cdot P^{-1} \oplus e \cdot P^{-1}$$

so it is $c_0 \cdot P^{-1} \oplus e'$ where error vector e' has weight t

Pros and cons

- "quantum resistant" – not to be broken by Shor algorithms (like RSA, DL)
- long studied (weak variants broken long time ago...)
- related to hard computational problems (Knapsack, LPN)

Cons:

- size
- use of randomness, an opportunity for covert channels