Public Key Cryptography: Encryption, Signatures, FDH

The ROM, FDH, using the ROM

FROM PREVIOUS LECTURE

- Ciphers
  - Stream ciphers : many follow OTP + PRG strategy
  - Block ciphers : work on plaintext of limited size = block
    output ciphertexts of same size
  - Modes of operation : used to encrypt longer messages

- Hash functions
  - Basic properties : first/second preimage resistance, collision resistance
  - Can be used to construct primitives like HMacs

Part I
Background

Divisors, Primes, GCD

- Assume: positive integers $a, b \in \mathbb{N}$
- Division: "a divides b" iff. $\exists\, k \in \mathbb{N}$ s.t. $a = k \cdot b$
  - We write $a \mid b$ and say a is a divisor of b
- Examples: $2 \mid 24$, $11 \mid 121$, etc.
- Prime numbers: positive integers greater than 1 only divisible by 1 and themselves
  - 1 is not a prime number. Nor is 0.
- Modular arithmetic: remainder of division
  - $a \bmod b = r$ s.t. $\exists\, k \in \mathbb{Z}$ with $a = kb + r$ and $r \in \mathbb{N}$
  - E.g. $15 \bmod 2 = 1$; $235 \bmod 5 = 0$; $135 \bmod 11 = 3$

Equivalence Classes, GCD

- Equivalence mod n:
  - $a \cong_n b$ iff. $a \bmod n = b \bmod n$
- Equivalence classes a_n:
  - $a_n = \{b \in \mathbb{Z} \mid a \cong_n b\}$
  - For instance $3_{12} = \{\ldots - 12, 3, 15, 27, \ldots\}$

- Common divisor: d is common divisor of a, b iff.:
  - $d \mid a$ and $d \mid b$
- Greatest common divisor: largest such d
  - $\mathrm{GCD}(15{,}35) = 5$
  - $\mathrm{GCD}(52, 236) = 4$

Finding GCD

- If $a \geq b$, it holds that: $\mathrm{GCD}(a, b) = \mathrm{GCD}(b, a \bmod b)$
  - This is because if $d \mid a$ and $d \mid b$, then $d \mid (a \bmod b)$
  - Why? Write $a = bq + r$, $a = kd$, $b = sd$

    Then $kd = qsd + r$, so $d(k - qs) = r$ and $d \mid r$
- For any $a \geq b$: if $a \bmod b = 0$ then $\mathrm{GCD}(a, b) = b$

- Hence Euclid's algorithm, input $a \geq b$:
  - 1. if $a \bmod b = 0$, then output b
  - 2. else, repeat procedure on input $(b, a \bmod b)$
- Total complexity: $O(\log^2 a)$

Extended GCD

- Theorem:
  - If $d = \text{GCD}(a, b)$, then d is the smallest positive integer for which there exist integers $r.s$ such that:
$$d = ar + bs$$
- If $d = 1$, a, b are called co-prime

- Extended GCD:
  - Input a, b
  - Output: d, r, s

Groups

- Set $\mathbb{G}$, operator $\circ$ such that:
  - Closure: $\forall\, a, b \in \mathbb{G}$ it holds $a \circ b \in \mathbb{G}$
  - Associativity: $\forall\, a, b, c \in \mathbb{G}$ it holds $(a \circ b) \circ c = a \circ (b \circ c)$
  - Identity element: $\exists\ \ e \in \mathbb{G},\ \ \forall a \in \mathbb{G}$ s.t.: $a \circ e = e \circ a = a$
  - Inverse element: $\forall a\ \ \exists a^{-1}$ s.t.: $a \circ (a^{-1}) = (a^{-1}) \circ a = e$

- $(\mathbb{G},\circ)$ is an Abelian group iff:
  - $(\mathbb{G},\circ)$ is a group
  - $\forall a, b \in \mathbb{G} : \quad a \circ b = b \circ a$

- Example: $(\{0, \ldots, n-1\}, +(\mathrm{mod}\, n)\ )$
  - Another example: $(\mathbb{Z},*\ \mathrm{mod}\ p)$

Subgroups and Orders

- Order $|\mathbb{G}|$ of group $(\mathbb{G},\circ)$: # elements in $\mathbb{G}$
- Subgroup $(\mathbb{H},\circ)$ of $(\mathbb{G},\circ)$:
  - $(\mathbb{H},\circ)$ is a group
  - $\mathbb{H} \subseteq \mathbb{G}$

- Theorem [Lagrange]:
  - If $\mathbb{G}$ is finite and $(\mathbb{H},\circ)$ subgroup of $(\mathbb{G},\circ)$
  - Then $|\mathbb{H}|$ divides $|\mathbb{G}|$

Cyclic Groups

- Cyclic groups $(\mathbb{G},\circ)$ of order n is cyclic iff.:

$$\mathbb{G} = \{g, g \circ g, \ldots, \underbrace{g \circ g \circ g \ldots \circ g}_{\text{n times}}\}$$

- We call g a generator of this group
- Any element can be a generator

- Theorem [Fermat's little theorem]:
  - If $(\mathbb{G},\circ)$ is a finite subgroup
  - Then $\forall\, a \in \mathbb{G}$ it holds that $a^{|\mathbb{G}|} = 1$

Groups and subgroups we use

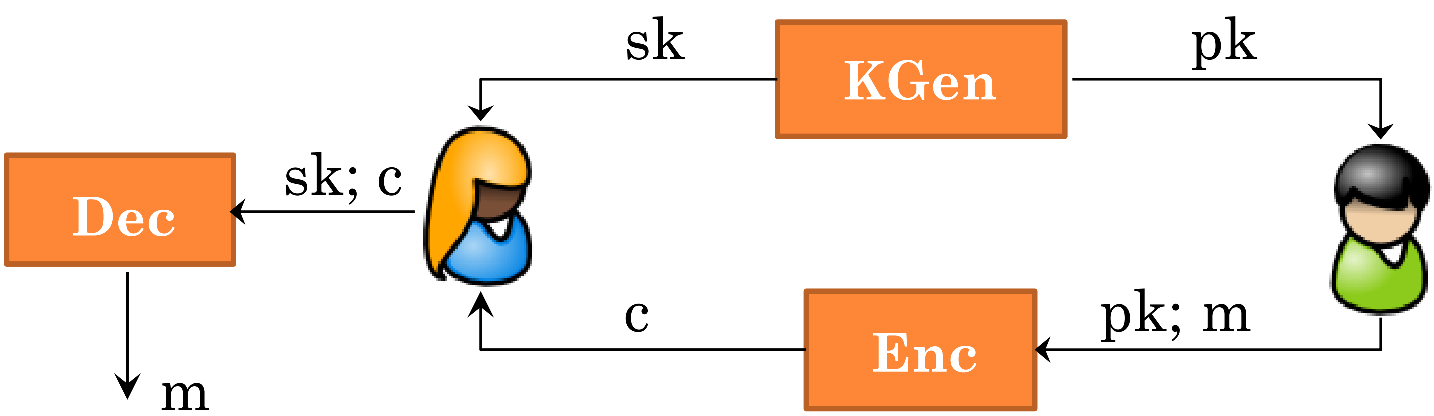
- For a prime p: $(\mathbb{Z}_p^*,\ *_{\text{mod } p})$
  - Integers modulo a prime, under multiplication mod p
  - Abelian (multiplication is commutative)
- Variation: sometimes in ECC we use $\left(E(\mathbb{Z}_{p^2}), +_E\right)$

- For primes p, q: $(\mathbb{G}, *_N)$ with $N = pq$
  - $\mathbb{G} = \{1 \leq g \leq N-1 \text{ s.t. } \mathrm{GCD}(g, N) = 1\}$
  - Cardinality: # of numbers co-prime with N
    - Usually denoted by Euler's Φ function:
    - $\Phi(pq) = (p-1)(q-1)$
  - E.g.: $p = 3; q = 7;\ \mathbb{G} = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$

Part II
Encryption Schemes

PUBLIC-KEY ENCRYPTION

- Syntax: algorithms $(\mathrm{KGen}, \mathrm{Enc}, \mathrm{Dec})$ such that:
  - $\mathrm{KGen}(1^\lambda)$: given security parameters, outputs tuple (sk, pk) consisting of a private/public key
  - $\mathrm{Enc}(pk; m)$: given plaintext and public key, outputs ciphertext c
  - $\mathrm{Dec}(sk; c)$: given ciphertext and secret key, outputs plaintext $\hat{m}$ or error symbol $\perp$



PUBLIC-KEY ENCRYPTION

- Correctness:
  - For all tuples $(sk, pk) \leftarrow$ KGen(1^λ) and for all plaintexts $m \in \mathbb{M}$, it must hold that $Dec\big(sk;\ Enc(pk; m)\big) = m$
  - Sometimes we degrade it to ϵ-correctness in which the decryption fails with probability ϵ
- IND-CPA: eavesdropper can't tell even 1 bit of p-text

$$(sk, pk) \leftarrow \text{KGen}\ (1^\lambda)$$
$$b \leftarrow_\$ \{0,1\}$$
$$(m_0\ , m_1) \leftarrow \mathcal{A}\ (pk, 1^\lambda)$$
$$c \leftarrow \text{Enc}(pk; m_b)$$
$$d \leftarrow \mathcal{A}(c, pk, 1^\lambda)$$

$\mathcal{A}$ wins iff. $d = b$

El-Gamal Encryption

- Before key-generation: setup
  - Pick primes p, q such that $p = 2q + 1$
  - Group $\mathbb{H} = (\mathbb{Z}_p^*, *_{\bmod p})$ and cyclic subgroup $\mathbb{G}$ of $\mathbb{H}$ of prime order q under the same operation
  - Generator g of $\mathbb{G}$

- Key generation:
  - Secret key $sk \leftarrow_\$ \{1, \ldots, q-1\}$; public key $pk = g^{sk} \bmod p$
- Encryption of message $m \in \mathbb{G}$:
  - Pick $r \leftarrow_\$ \{1, \ldots, q-1\}$, set $c = (g^r \bmod p,\ m \cdot pk^r \bmod p)$
- Decryption of $c = (c_1, c_2)$:
  - Set $\hat{m} = {}^{c_2}/_{c_1^{sk}}$

Generic Messages

- Message has to be in $\mathbb{G}$
- What happens otherwise?
  - Could use m^2, for $m \in \mathbb{H} \setminus \mathbb{G}$ (if $m \in \mathbb{H} \setminus \mathbb{G}$, then the order of m is not q; yet, the order of m^2 is q) **Proof in TD**
    - Encrypt m^2 instead of m, take $\sqrt{\hat{m}}$ at decryption
  - Could also modify scheme a little bit, using a hash function:
    - Encryption: $(g^r, H(pk^r) \oplus m)$
    - Decryption: $\hat{m} = c_2 \oplus H(c_1^{sk})$
    - We can prove security as long as the hash function H preserves the pseudorandomness of pk^r

El-Gamal Security

- Theorem:
  - If there exists an adversary $\mathcal{A}$ who can break the IND-CPA security of the El Gamal scheme with probability $\frac{1}{2} + \mathrm{Adv}_{\mathcal{A}}$…
  - … then there exists an adversary $\mathcal{B}$ who can break the DDH assumption in group $\mathbb{H}$ with probability $p_{\mathcal{B}}$ such that:

$$p_{\mathcal{B}} = \frac{1}{2} + \frac{1}{2}\mathrm{Adv}_{\mathcal{A}}$$

Reminder: Hard problems based on DLog

- Setup:
  - Cyclic group $\mathbb{G}$ of prime order q, generator g
- DLog:
  - Given q, g, g^a, find $a \in \{1, \dots q-1\}$ (g and q fully define $\mathbb{G}$)
- CDH
  - Given q, g, g^a, g^b find g^{ab}
- DDH
  - Given q, g, g^a, g^b, g^c find out whether $c = ab$ or not

- Note:
  - If DLog is solved, then we can solve CDH
  - If we can solve CDH, then we can solve DDH

Proof

- What does breaking DDH mean?
- B plays a game against a challenger
  - Depending on a bit b, B receives $\left(g, g^a, g^b, g^{ab}\right)$ (if $b = 1$) or (g, g^a, g^b, g^c), for $a, b, c \leftarrow_\$ \{1, \ldots q\}$
  - B must output a bit guess_B and wins iff. $\text{guess}_B = b$

- Constructing B that uses A
  - Upon receiving tuple $\left(g, g^a, g^b, g^z\right)$ with $z = ab$ or $z = c$
  - B gives A: $(g, pk = g^a)$
  - A chooses and sends B messages (m_0, m_1)
  - B chooses a bit b^*, outputs $(g^b, g^z \cdot m_{b^*})$, send to A
  - A outputs guess_A and wins iff $\text{guess}_A = b^*$
  - B outputs $(\text{guess}_A == b^*)$

Analysis

- Constructing B that uses A
  - Upon receiving tuple $\left(g, g^a, g^b, g^z\right)$ with $z = ab$ or $z = c$
  - B gives A: $(g, pk = g^a)$
  - A chooses and sends B messages (m_0, m_1)
  - B chooses a bit b^*, outputs $(g^b, g^z \cdot m_{b^*})$, send to A
  - A outputs guess_A and wins iff $\text{guess}_A = b^*$
  - B outputs $(\text{guess}_A == b^*)$

- Analysis:
  - If b = 1, B got $\left(g, g^a, g^b, g^{ab}\right)$, which means A plays the true game: so A wins w.p. $\frac{1}{2} + \text{Adv}_A$
  - If b = 0, B got $\left(g, g^a, g^b, g^c\right)$, so A wins w.p. $\frac{1}{2}$

Malleability

- Malleability, to maul:
  - Informally: ability to "re-shape" things
  - Not always bad – crucial in homomorphic crypto
  - Bad for IND-CCA

- ElGamal is malleable:
  - Say we encrypt message m with randomness r
    $$(c_1, c_2) = (g^r, m \cdot pk^r)$$
  - Now pick random $s \leftarrow_\$ \{1, \ldots, q-1\}$
  - Maul ciphertext: $c_1^* = c_1^s = g^{rs}$, $c_2^* = c_2^s = m^s\, pk^{rs}$
  - Then (c_1^*, c_2^*) is an encryption of m^s

IND-CPA VS IND-CCA

- IND-CPA: eavesdropper can't tell even 1 bit of p-text

$$(sk, pk) \leftarrow \text{KGen}\,(1^\lambda)$$
$$b \leftarrow_{\$} \{0,1\}$$
$$(m_0\,, m_1) \leftarrow \mathcal{A}\,(pk, 1^\lambda)$$
$$c \leftarrow \text{Enc}(pk; m_b)$$
$$d \leftarrow \mathcal{A}(c, pk, 1^\lambda)$$

$\mathcal{A}$ wins iff. $d = b$

- IND-CCA: even if we have power of decryption, can't learn even 1 bit of fresh message
  - Same as before, but include Dec. oracle
  - A must not query challege ciphertext to Dec.

Malleability and IND-CCA

- Malleability informally means that one can use a relation on the input to induce a relation on the output.

- Malleability usually implies encrpytion scheme is not IND-CCA
- Why?
  - Key to IND-CCA success: A cannot query the challenge ciphertext
  - Maul challenge ciphertext, then query it to Dec
  - Perform inverse transformation

IND-CCA ENCRYPTION

- Much harder to get than IND-CPA encryption
- Must prevent malleability, so usually we would use something to verify the integrity of the message

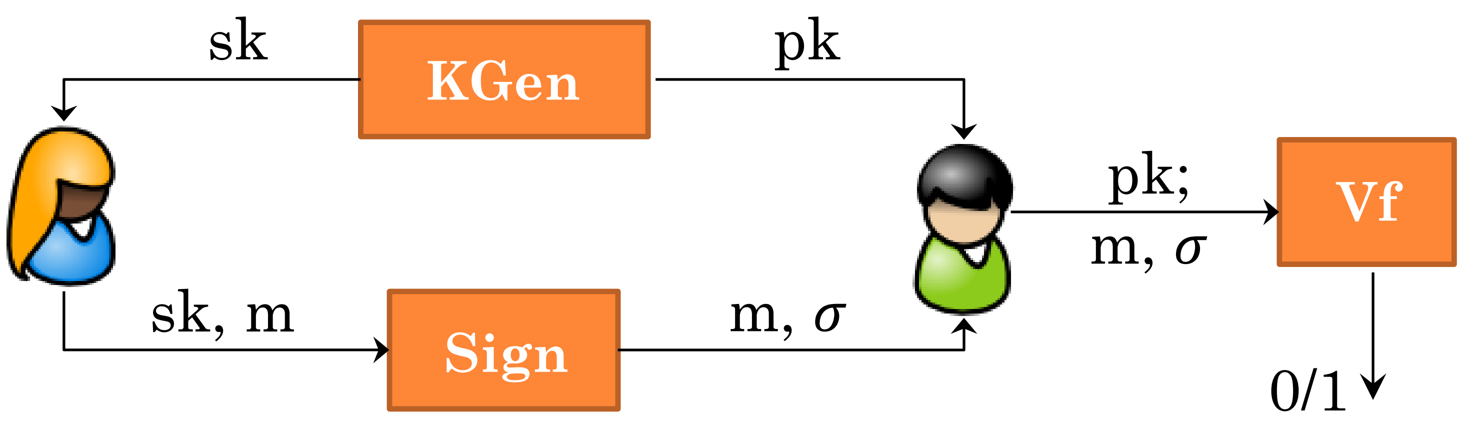
- Would using a hash function help?
  - $\text{Enc}(pk, H(m))$: doesn't work. **Why not?**
  - How about $H(\text{Enc}(pk;\ m))$?

- Could we use a PRF instead?
  - $\text{Enc}(pk, \text{PRF}(K, m))$: security is ok, but why would we do PKE if we already had a shared key?

Part III
Signature Schemes

DIGITAL SIGNATURES

- Syntax: algorithms $(\mathrm{KGen}, \mathrm{Enc}, \mathrm{Dec})$ such that:
  - $\mathrm{KGen}(1^\lambda)$: given security parameters, outputs tuple (sk, pk) consisting of a private/public key
  - $\mathrm{Sign}(sk; m)$: given plaintext and secret key, outputs signature σ
  - $\mathrm{Vf}(pk; m, \sigma)$: given message, signature and public key, outputs a bit 1 if σ checks for m, 0 otherwise



Signature security

- Correctness:
  - For all tuples $(sk, pk) \leftarrow \text{KGen}(1^\lambda)$ and for all messages $m \in \mathbb{M}$, it must hold that $\text{Vf}\big(pk; m, \text{Sign}(sk; m)\big) = 1$
  - Sometimes we degrade it to ϵ-correctness in which the verification of a signed message fails with probability ϵ
- EUF-CMA: adversary can't forge fresh signature

$$(sk, pk) \leftarrow \text{KGen}\ (1^\lambda)$$

$$(m\ , \sigma) \leftarrow \mathcal{A}^{\text{Sign}(*)}\ (pk, 1^\lambda)$$

Store list $\mathbb{Q} = \{(m_1, \sigma_1), \ldots (m_k, \sigma_k)\}$ of queries to Sign

---

$\mathcal{A}$ wins iff. $(m,\ *) \notin \mathbb{Q}$ and $\text{Vf}(pk; m, \sigma) = 1$

RSA Signatures

- RSA setup:
  - Large primes p, q, let $N = pq$
  - Subgroup of co-primes with N, size $\Phi(N) = (p-1)(q-1)$
  - Work in subgroup mod $\Phi(N)$
- RSA signatures:
  - KGen: Find $e \in_R \{1, \dots, \Phi(N)\}$ such that $\mathrm{GCD}(1, \Phi(N))$ and its inverse d such that $e \cdot d = 1\ mod\ \Phi(N)$
    - Public key $PK = (N, e)$; Secret key $SK = d$
  - Sign message m:
    - $\sigma = m^d \bmod N$
  - Verify signature σ for message m
    - Output 1 iff. $m = \sigma^e \bmod N$ and output 0 otherwise

Not EUF-CMA

RSA Signature

- Key Generation:

$\boxed{pk = N, e}$ $\boxed{sk = d}$

- Sign:

$$\boxed{\sigma = m^d \ mod\ N}$$

- Verify:

$$\boxed{m \stackrel{?}{=} \sigma^e \ mod\ N}$$

- No Sign(·) queries:
  - Pick random string s
  - Compute $\hat{m} = s^e \bmod N$
  - Output $(\hat{\boldsymbol{m}}, \boldsymbol{s})$ as forgery
- Forgery with 2 queries:
  - Want to forge signature for given message m
  - Pick m_1 at random, ask signature: $\sigma_1 = m_1^d \bmod N$
  - Compute m_2 s.t. $m_1 m_2 = m \bmod N$, get $\sigma_2 = m_2^d \bmod N$
  - Output $(\boldsymbol{m}, \boldsymbol{\sigma_1 \sigma_2} \ \mathbf{mod}\ \boldsymbol{N})$

How to Get EUF-CMA

- Use Hash functions, and sign hash of message
- The Probabilistic Full-Domain-Hash RSA scheme:
  - Use a hash function $H: \{0,1\}^* \to \mathbb{Z}_N^*$
  - KGen: Obtain $(N, e, d) \leftarrow \mathrm{KGen}_{RSA}(1^\lambda)$, set:
    $$PK = (N, e);\ \ SK = d$$
  - Sign: Choose random $r \in_\$ \{0,1\}^*$, compute $y = H(r\,||\,m)$, output signature:
    $$\sigma = (r, y^d \bmod N)$$
  - Verification: receive $m, \sigma = (r, s)$, output 1 iff.
    $$s^e = H(r\,||\,m)$$

Security of PFDH-RSA

- Assumptions on hash functions:
  - Collision-resistance sometimes suffices
  - However, proofs for signatures are hard to do relying just on collision resistance
  - Need a stronger assumption

- Random oracles, the ROM:
  - Imagine an idealization of a hash function
  - Every time we query the idealization on a value x, check RO has not been queried with x before:
    - If so, output new uniformly random value of good length
    - Else output previously seen value for x

RSA ASSUMPTION

- The RSA problem:
  - Given an RSA instance, with public key (N, e)
  - Given "ciphertext": $C = m^e \bmod N$
  - Compute m

- The RSA assumption:
  - The RSA problem is hard to solve for a PPT adversary

- The strong RSA assumption:
  - Alow Adversary to choose exponent e
  - Given (N, C), hard to output (m, e) s.t. $C = m^e \bmod N$

Security of PFDH

- Theorem:
  - Take $|r| = \mathrm{Log}\, q_S$
  - In the random oracle model
  - If there exists an adversary A against the EUF-CMA of the PFDH scheme, making at most q_H queries to H and at most q_S queries to Sign, winning with probability p_A…
  - Then there exists an adversary B that solves the RSA problem with probability

$$p_B \geq \frac{1}{4} p_A$$

Programming a RO

- Key observations:
  - A does not have much use submitting messages to Sign oracle without submitting them to Hashing RO first
    - Not entirely true, we would lose a guessing term here
  - A cannot output a meaningful forgery for a message m without submitting it to Hashing RO first
    - Again, not entirely true, same considerations as before
  - A has no use querying the same message twice to the random oracle (since the RO always returns the same thing)

Security Proof for PFDH-RSA

- Proof intuition:
  - The random oracle randomizes the messages to be signed; in fact, by choosing different values of r we get different values of $H(r \,||\, m)$
  - Multiple related signatures per message:
    - $m \xrightarrow{r_1} (r_1, [H(r_1 \,||\, m)]^d \mod N)$
    - $m \xrightarrow{r_2} (r_2, [H(r_2 \,||\, m)]^d \mod N)$
    - … … … … … …
    - $m \xrightarrow{r_k} (r_k, [H(r_k \,||\, m)]^d \mod N)$
  - Because of the RO, all hashes are different

Constructing the Reduction

- Adversary B plays the RSA problem
- It needs to simulate the EUF-CMA game to adversary A, and use its output

- Setup:
  - Adversary B receives tuple (N, e) and $C = m^e \bmod N$ for some m
  - B must then answer queries from A for signatures
  - B prepares for each m a list of q_S values like this:
    - Choose random r_i
    - Choose random $x_i < N$
    - Given e calculate: $z_i = x_i^e$
    - Store tuple (m, r_i, x_i, z_i); all tuples with same m make up L_m

THE REDUCTION

- Every time A queries the RO $H(m||r)$, B responds as follows:
  - Create initially empty table $\mathbb{T}$ with entries $(\cdot,\cdot,\cdot)$
  - If m is queried for the first time, B first makes up L_m
  - Else, assume L_m is already created
  - If there exists in $\mathbb{T}$ an entry $(m||r, x, z)$, return z
  - If $r \in \{r_1, \ldots, r_k\}$ from list L_m, then output z_i and insert in $\mathbb{T}$ an entry $(m||r_i,\ x_i,\ z_i)$
  - Else, if r not used in L_m, choose random x and output to A the value $z = C\, x^e \ mod\ N$ and store $(m||r, x, z)$ in $\mathbb{T}$

- Remember A has q_S signature queries

Finishing the Reduction

- Apart from RO queries, A can ask signature queries to the signing oracle
  - B has to respond to these queries

- When A queries Sign(m):
  - If m does not have a corresponding L_m, generate it
  - Else, pick the next value of r in that list, see if there is a related entry $(m \,||\, r, x, z)$ in $\mathbb{T}$, output (r, x)
  - If there is no such related entry, create one, and output the same thing

Winning or Losing

- Finally A outputs a forgery of the type:

$$(m, (r, s))$$

  - If $r \in L_m$, abort
  - Else, if $r \notin L_m$, find corresponding entry in $\mathbb{T}$ and output (to B's challenger):

$$\frac{s}{x} \mod N$$

- Note: A outputs forgery on message not queried to signature oracle before
  - But he could have input $(m \mid\mid r)$ to RO instead, got x
  - Only way to get r from L_m is by guessing it:

Total probability it doesn't happen: $(1 - 2^{-|r|})^{qs}$

Random Oracles

- Idealising hash function in a very useful way
  - Can get nice properties for key-exchange, encryption, signatures, and many other primitives
- However, random oracles are a bit too ideal
  - We know that some primitives that are "secure" in the presence of random oracles are insecure no matter which hash function we use for our RO
- Proofs in ROM:
  - Tricky bit is to program the RO: store queries, know what to answer
- Alternative to ROM: standard model

Full-Domain Hashing

- Generalized beyond RSA by trapdoor permutations

- Trapdoor permutations:
  - Family of 1-way permutations $\{f_K: D_k \to R_k\}$ with $K \in \mathbb{K}$, such that $D_K, R_k, \mathbb{K}$ are binary sets of arbitrary length. Includes algorithms $(\text{Gen}, \text{Sample}, f, f^{-1})$ such that:
    - Gen: on input 1^λ outputs tuple $K \in \mathbb{K}$ and trapdoor T
    - Sample: on input the key K, this algorithm efficiently samples input $x \in D_K$
    - f: on input K and any $x \in D_K$, efficiently outputs $\text{y} = f_K(x)$
    - f^{-1} : on input K, trapdoor T and any $y \in R_K$, efficiently outputs inverse x such that $y = f_K(x)$
    - Security: without trapdoor T, hard to invert f

PKE as Trapdoor Permutation

Trapdoor permutation

- Algorithm Gen

$\boxed{K}$ $\boxed{T}$

- Function f: efficient to get

$$\boxed{y = f_K(x)}$$

- Inverse f^{-1} easy with T

$$\boxed{x = f_K^{-1}(T, y)}$$

PKE

- Algorithm KGen

$\boxed{PK}$ $\boxed{SK}$

- Encryption algorithm

$$\boxed{y = \mathrm{Enc}_{PK}(x)}$$

- Decryption algorithm

$$\boxed{x = Dec_{SK}(y)}$$

GENERALIZED FDH

- Take Trapdoor permutation $\text{TDP} = \{\text{Gen}, \text{Sample}, f, f^{-1}\}$
- Take hash function $H: \{0,1\}^* \to \{0,1\}^n$

- Key Generation: Run $(K, T) \leftarrow \text{Gen}(1^\lambda)$. Set:

$$PK \coloneqq K \quad \text{and} \quad SK = T$$

- Signing: Compute $r \coloneqq H(m)$, then do: $y \coloneqq \text{Sample}\,(PK; r)$

$$\text{Signature is: } \sigma = f_T^{-1}\,(y)$$

- Verification: Do $r \coloneqq H(m)$, then: $y \coloneqq \text{Sample}\,(PK; r)$.

$$\text{Output 1 iff. } f(\sigma) = y$$