CS4236 Cryptography Theory and Practice Topic 13 - Secret sharing and MPC

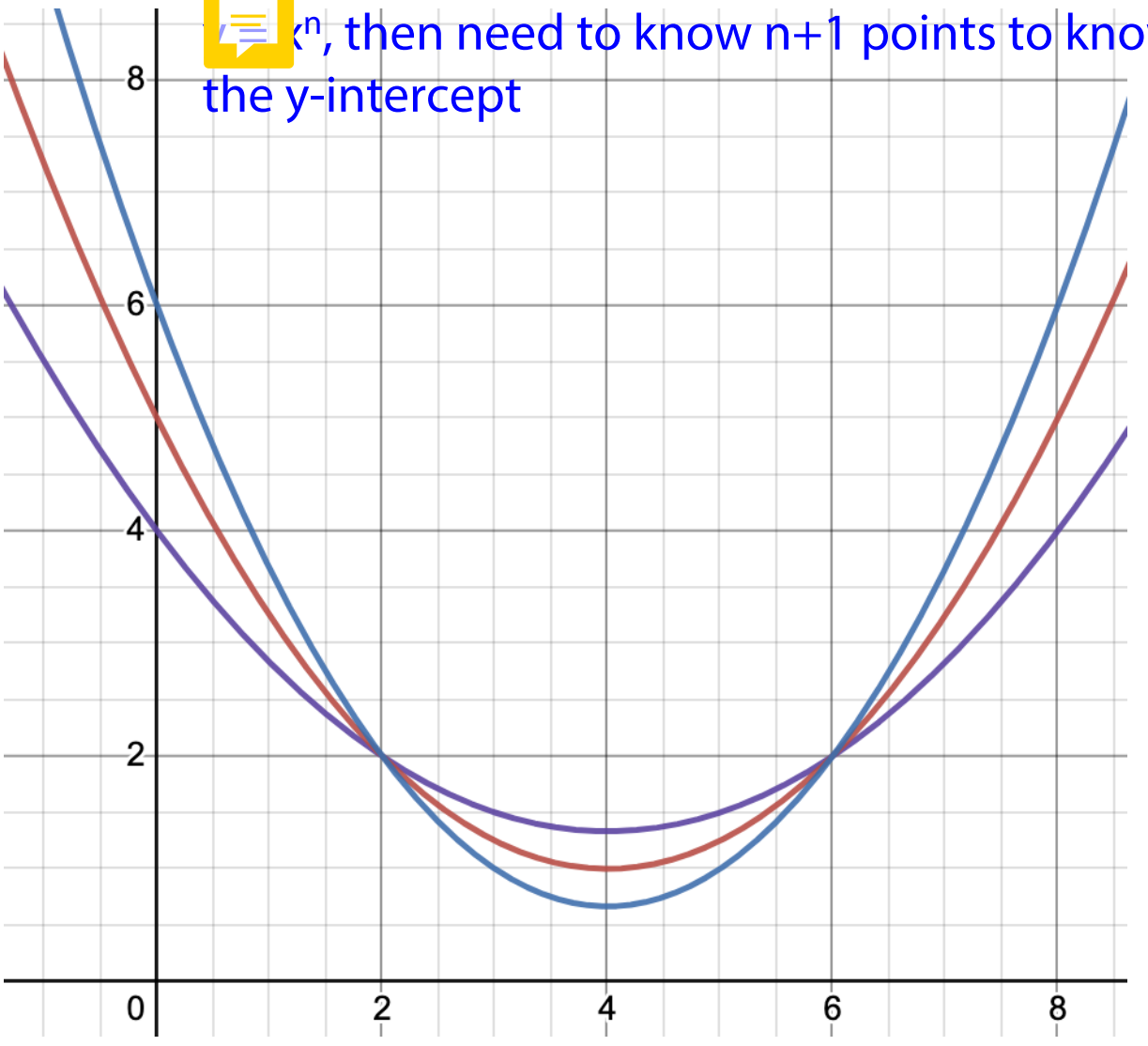
Hugh Anderson

National University of Singapore
School of Computing

November, 2022

Secret sharing...

the point is to show that for a degree n graph $y = x^n$, then need to know n+1 points to know the y-intercept



Outline

1. **Secret sharing**
   - Secret Sharing (SS)
   - Secure multi-party computation (MPC)

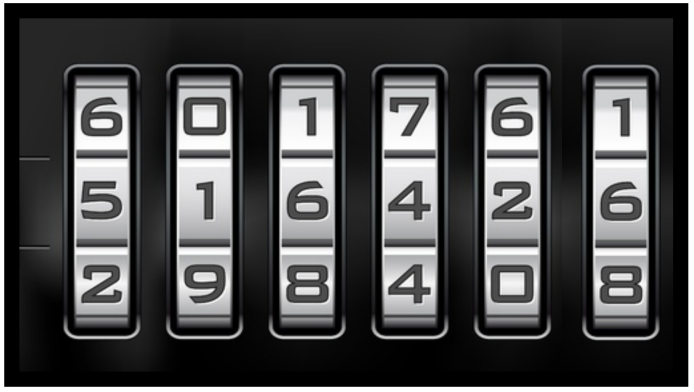
2. **Trying to put everything in place**
   - Components/structures in the course...
   - Ideas/structures, algorithms, constructions, applications
   - Cryptanalysis. Attacks. And more attacks. And more...
   - The exam topics

The story so far ... where have we been?

The last 12 weeks: weekly steps we have taken

1. We had a historical/contextual introduction in Session 1.
2. We progressed from perfect *secrecy* to perfect *indistinguishability*.
3. *Perfectly* indistinguishable was relaxed to give *computationally* indistinguishable. An EAV-Secure system was constructed with a PRG.
4. The notion of CPA-secure was developed, and achieved with a PRF.
5. Modes, the “padding oracle”, and CCA-Security were outlined.
6. We looked at cryptographic MACs and *authenticated* encryption.
7. We began looking at hashes, with definitions, games, and applications.
8. We then looked at birthday attacks on hashes, and rainbow tables.
9. We looked at SP networks, and cryptanalysis of them.
10. We looked at definitions of PK systems, and the RSA construction.
11. We looked at Key exchange, Elgamal and ECC.
12. Last week, we looked at digital signatures, hash-and-sign, Elgamal sigs, and side-channel attacks on cryptography.

Motivating example for sharing a secret



Are bank managers trustworthy? **Ch 13.3, pg 501**

Imagine that a bank has a vault that can be opened using a secret pin k. The secret k is to be kept by 3 managers. Only when three of them get together can the vault be opened. How should the secret k be shared among the three managers?

Here is a simple method. Suppose k is a 6 decimal digits sequence. The dealer splits it into 3 equal parts, i.e $k = s_1 + s_2 + s_3$. Manager_i will get the "share" s_i. Hence, each manager will keep 2 digits.

Question: Suppose Manager_1 and Manager_2 are malicious. They combine their shares and try to open the vault. How many combinations do they have to try in the worst case?

Definition of a secret sharing scheme

Definition $\mathrm{SS} = (\mathrm{Init}, \mathrm{Share}, \mathrm{Reconstruct})$

A dealer holds a secret that it wishes to share among some set of w participants $\mathcal{P} = \{P_1, \ldots, P_w\}$ by giving each a share. Any t participants should be able to pool their shares and reconstruct the secret.

$\mathrm{Init}(n, w)$**:** The dealer chooses and initializes the SS scheme. n is the security parameter, w is the number of participants.

$\mathrm{Share}(k, t, \mathcal{P})$**:** Input is a secret k, and t, the number of participants needed to reconstruct the secret. The output is $\mathcal{S} = \langle s_1, \ldots, s_w \rangle$, the shares for each participant in $\mathcal{P}$: $\mathcal{P} \leftarrow \mathrm{Share}(k, t, \mathcal{P})$

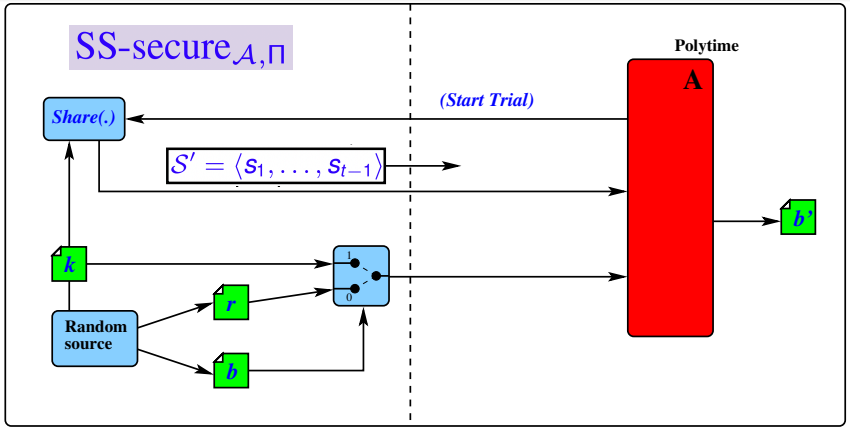
$\mathrm{Reconstruct}(S)$**:** Returns key k (from at least t shares): $k \leftarrow \mathrm{Reconstruct}(S)$

See pg 501 for details.

Secret sharing properties...

I will only show one formal definition of a security property for secret sharing schemes, but you might be concerned with other properties such as leakage of keys and shares.

Property for SS: Collusion resistance



The game: SS-secure$_{\mathcal{A},\Pi}$ (cannot collude)

1. Adversary can do multiple trials, where the challenger generates new shares $\mathcal{S}$ for a new secret k. In each trial, the adversary is given all except one of the shares $\mathcal{S}' = \langle s_1, \ldots, s_{t-1} \rangle$, and also (through a random bit b) either k or a randomly generated r. Adversary outputs b'.
2. Adversary wins where $b = b'$ and the output is SS-secure$_{\mathcal{A},\Pi} = 1$.

Definition: Perfect (unconditional) collusion resistance

An SS scheme Π is perfectly (i.e. unconditionally) collusion resistant, or SS-secure, if for all PPT adversaries $\mathcal{A}$:

$$\Pr[\text{SS-secure}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}$$

A better way to split/share the secret...

Construction #1: Use a hidden sum policy...

We will use the definition of the SS scheme to show this construction.

$\text{Init}(10^6, 3)$**:** There are 3 participants in total.

$\text{Share}(k, 3, \mathcal{P})$**:** Randomly pick the two keys s_1, s_2, from $\mathbb{Z}_{10^6}$, and then compute $s_3 = (k - s_1 - s_2) \bmod 10^6$. Send each share s_i to the managers: $P_1, P_2, P_3 \leftarrow s_1, s_2, s_3$

$\text{Reconstruct}(S)$**:** Returns key k: $k = (s_1 + s_2 + s_3) \bmod 10^6$

Now, suppose P_1 and P_2 get together and combine their shares. With the knowledge of s_1, s_2, what are the number of possible k values?

This construction is (provably) unconditionally collusion resistant. There is no assumption of hardness.

Useful for multi-party protocols

Secret sharing schemes are useful in some protocols that involve multiple parties. For example, a method for a few parties to "collectively" generate a pair of RSA private and public keys such that no single party knows the whole private key (the actual construction is quite involved).

Threshold secret sharing

Threshold schemes have $t < w$

Consider an additional requirement: any 2 of the 3 managers can open the vault, but a single manager is unable to open the vault. Schemes meeting such a requirement are called threshold schemes (if $t = w$ it is just a sharing scheme).

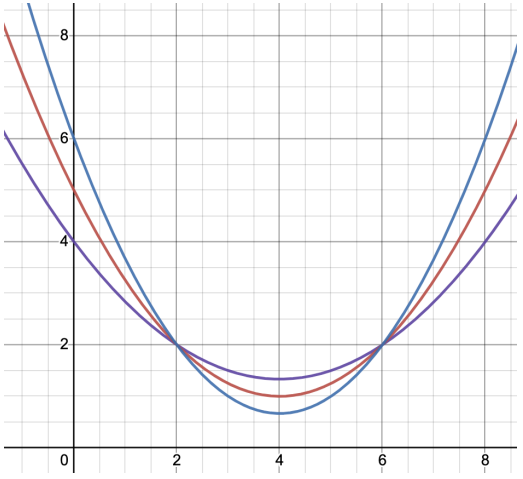
Here is an example of such a situation in the real-world: the control of nuclear weapons in Russia depended upon a “2-out-of-three” mechanism. The 3 parties involved were, the President, the Defense Minister and the Defense Ministry.

Definition of a (t, w)-threshold scheme

A (t, w)-threshold scheme is an SS scheme with $t < w$.

With this scheme any t participants can compute the value of k, but no group of $t - 1$ participants can do so.

Construction of a (t, w)-threshold scheme



w participants ie w=8 but t=3

Intuition of Shamir's (t, w)-threshold scheme (1979)

The main idea is based on having sufficient points to fully define a polynomial curve. The dealer randomly picks a polynomial f of degree $t-1$ s.t. $f(0) = k$. The polynomial is over a finite field, say $\mathbb{Z}_p$, and each coefficient is uniformly chosen from $\mathbb{Z}_p$. Participant P_i gets the value $f(i)$.

When t participants get together, they have t samples of the polynomial f. With t samples, they can reconstruct the $t-1$ degree polynomial f and thus get the secret $f(0)$.

Construction #2: Shamir's (t, w)-threshold scheme

Reconstruction of the curve using polynomial interpolation

Init$(1^n, w)$: The dealer chooses a large prime p and makes it public.

Share$(k, t, \mathcal{P})$: The dealer randomly chooses $t-1$ elements a_i from $\mathbb{Z}_p$ for the curve $f(x) = (a_{t-1}x^{t-1} + \ldots + a_1 x + k) \bmod p$. The dealer computes $s_i = f(i)$ for $i = 1, 2, \ldots, w$, and securely sends s_i to P_i: $P_1, P_2, \ldots, P_w \leftarrow s_1, s_2, \ldots, s_w$

Reconstruct(S): Given any set of t shares, use the linear combination

$$k = f(0) = \left(\sum_{j=1}^{t} s_{i_j} \ell_j(0) \right) \bmod p$$

of Lagrange basis polynomials: 2 nested for loops

$$\ell_j(0) = \left(\prod_{1 \le k \le t, k \ne j} \frac{i_k}{i_k - i_j} \right) \bmod p$$

(These details of the use of Lagrange polynomials are not part of CS4236).

Construction #3: Verifiable (VSS) schemes

Construction of Feldman's (t, w)-VSS-threshold scheme

Suppose the dealer is dishonest. In a verifiable secret sharing scheme, in addition to sending the shares, the dealer publicly broadcasts extra elements to help others verify the shares are correct.

$\text{Init}(1^n, w)$: Dealer chooses a cyclic group $\mathcal{G}$ of prime order p where discrete log is hard, a generator g, and makes $\langle \mathcal{G}, g, p \rangle$ public.

$\text{Share}(k, t, \mathcal{P})$: The dealer randomly chooses t elements a_i from $\mathbb{Z}_p$ for the curve $f(x) = (a_{t-1}x^{t-1} + \ldots + a_1 x + a_0) \bmod p$. The key k is masked as $c = \mathcal{H}(a_0) \oplus k$. The dealer computes $s_i = f(i)$, securely sends s_i to P_i: $P_1, P_2, \ldots, P_w \leftarrow s_1, s_2, \ldots, s_w$

and also broadcasts extra elements: $\langle c, g^{a_0}, \ldots, g^{a_{t-1}} \rangle$

Note that g^{a_0} leaks a little information about a_0 (Legendre/DDH) and this is why the key is blinded.

$\text{Reconstruct}(S)$: Each participant can verify their s_i using the extra elements. If no participants have complaints, they can use their shares to find a_o and then compute $k = c \oplus \mathcal{H}(a_0)$

Verifiable (VSS) scheme

How to verify the shares?

Each participant P_i has their own share $s_i = f(i)$, and public elements $\langle \mathcal{G}, g, p \rangle$ and $\langle c, g^{a_0}, \ldots, g^{a_{t-1}} \rangle$.

The participant can compute

$$\prod_{j=0}^{t-1} (g^{a_j})^{i^j}$$

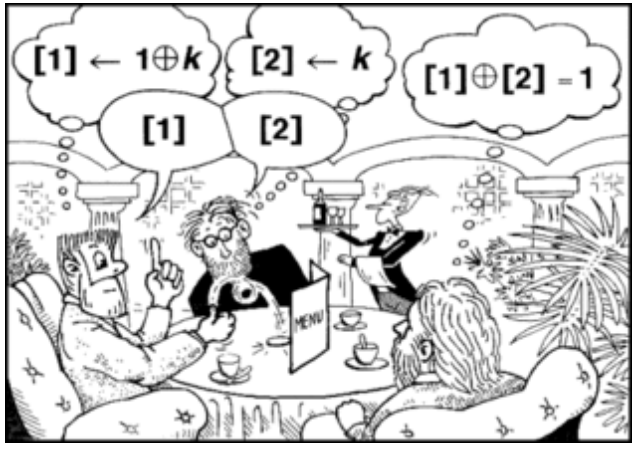
and check if it is equal to g^{s_i}. Why would this be so? Lets expand it:

$$\begin{aligned} \prod_{j=0}^{t-1} (g^{a_j})^{i^j} &= g^{\sum_{j=0}^{t-1} a_j \times i^j} \\ &= g^{f(i)} \\ &= g^{s_i} \end{aligned}$$

If the verify is incorrect the participant alerts the others over the public channel.

Computing a single bit value, anonymously

The Dining Cryptographers...



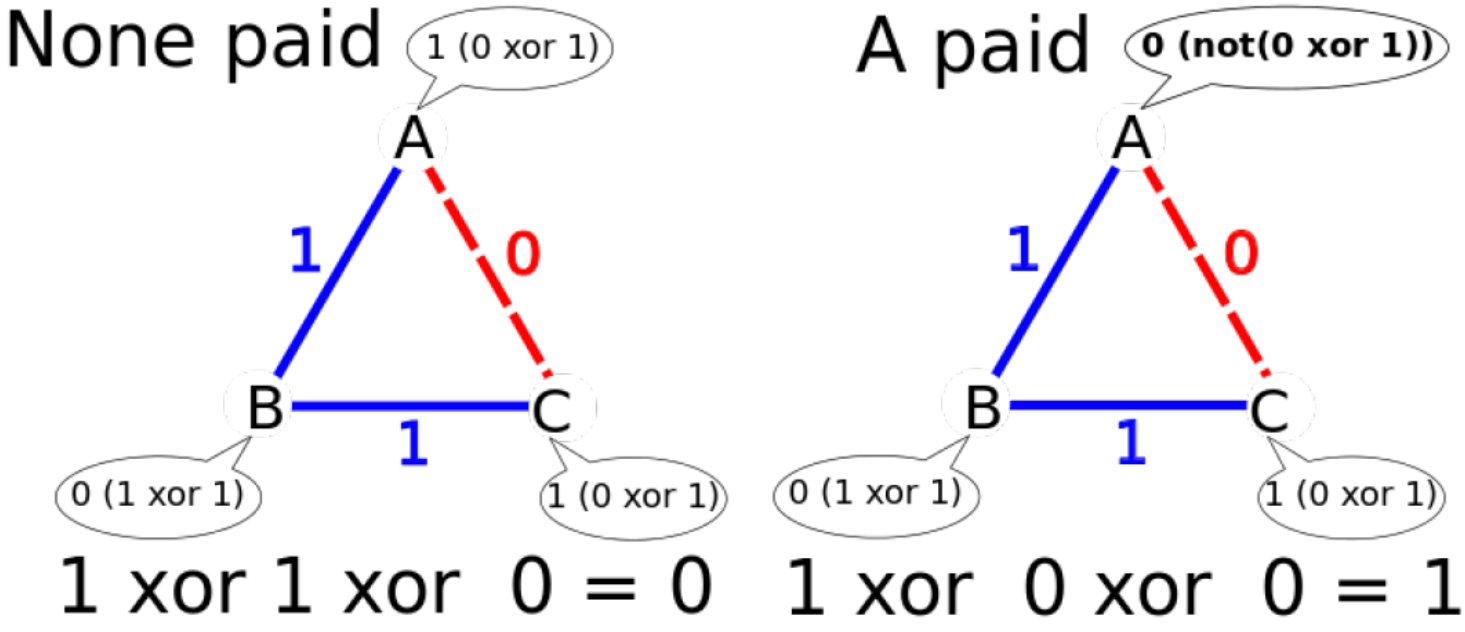
Consider a group of (3 or more) cryptographers, dining at a fancy restaurant. When they ask for the bill, the waiter tells them that the meal has been paid for, by someone who wishes to remain anonymous. The only other person who had been in the restaurant was from the NSA, so the cryptographers want to find out if the NSA guy paid, or if one of their own group paid.

Can they devise a protocol which lets them find out if one of their own group paid for the meal?

Anonymity protocol

The Dining Cryptographers...

Each pair of adjacent cryptographers decide on a randomly chosen bit (0/1) behind the menu (privately). If they did not pay for the meal, they announce the XOR of their two neighbours. Otherwise they announce the reverse.



The parity of the values is 1 if one of them paid for the meal.

Note that none of them know the other's "I paid" state.

Computing a more general function $\mathcal{F}(k_1, k_2, \ldots)$

MPC and sharing secrets

Suppose parties P_1, P_2, P_3 have secrets k_1, k_2, k_3, and wish to compute a function $\mathcal{F}(k_1, k_2, k_3)$, but without having to reveal their secrets. MPC is the protocol for computing $\mathcal{F}(k_1, k_2, k_3)$ for each of P_1, P_2, P_3 such that afterwards P_1 does not learn k_2, k_3, P_2 does not learn k_1, k_3, and P_3 does not learn k_1, k_2. An example from the world of ML might be where parties each have training data, and want the learnt model, but without revealing their data.

There is a theorem that shows that for any PPT $\mathcal{F}()$, there is a protocol for MPC, based on secret sharing.

Suppose that we have multiple secrets, $k_1, k_2, \ldots, k_m$, shared using the Shamir (t, w) scheme, with $s_{i,j}$ being the share of the secret k_i for P_j. We will only consider the two functions

$$\begin{aligned} \text{ADD}_p(k_1, k_2) &= (k_1 + k_2) \bmod p \\ \text{MUL}_p(k_1, k_2) &= (k_1 \times k_2) \bmod p \end{aligned}$$

because they can be used to form any computable function/circuit.

Definition of an MPC scheme

Definition $\text{MPC} = (\text{Init}, \text{Share}, \text{Eval}, \text{Reconstruct})$

Participants have secrets k_i, which they do not wish to reveal. However, they do wish to compute a composite function $\mathcal{F}(k_1, k_2, \ldots, k_w)$. Participants distribute shares of their secrets, evaluate the shares of the function $\mathcal{F}()$, and then reconstruct the value from the resulting shares.

$\text{Init}(n, \mathcal{P})$**:** A participant initializes the MPC scheme. n is the security parameter, $\mathcal{P}$ is the set of participants.

$\text{Share}(\mathcal{K})$**:** Input are the secrets $\mathcal{K}$. Output is $\mathcal{S} = \langle s_1, \ldots, s_w \rangle$, the shares for each participant in $\mathcal{P}$: $\mathcal{S} \leftarrow \text{Share}(\mathcal{K})$

$\text{Eval}(\mathcal{S})$**:** Input are the shares $\mathcal{S}$. The output is the shares of the function $\mathcal{F}_s = \langle f_{s_i}, \ldots, f_{s_w} \rangle$ $\mathcal{F}_s \leftarrow \text{Eval}(\mathcal{S})$

$\text{Reconstruct}(\mathcal{F}_s)$**:** Returns the final value. $f \leftarrow \text{Reconstruct}(\mathcal{F}_s)$

See paper.

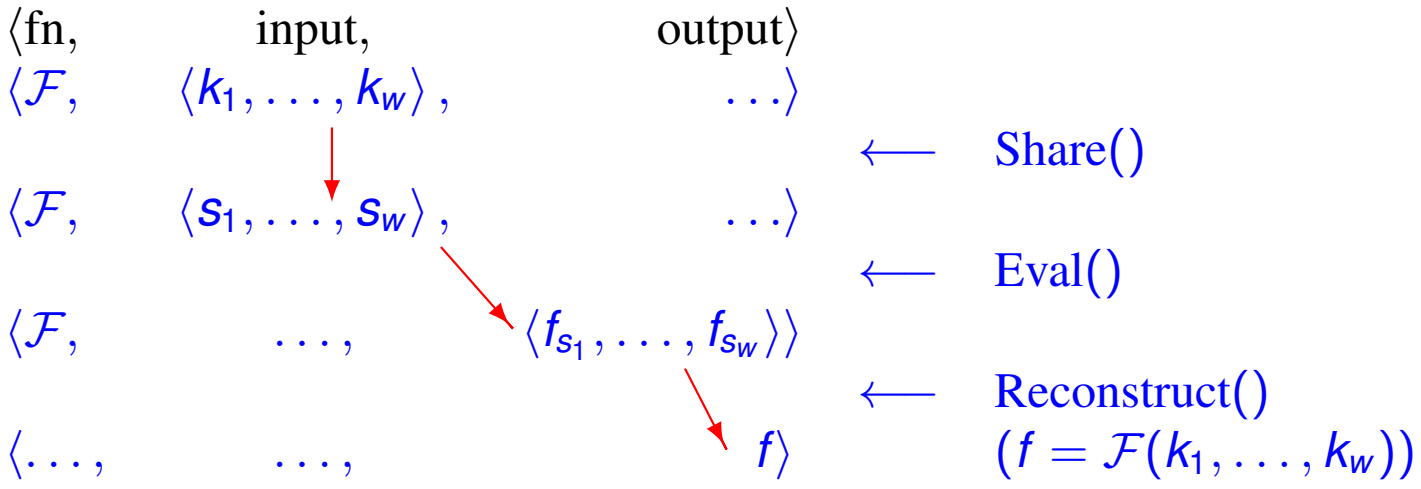
SS and MPC...

Note that Share and Reconstruct are the same ideas seen in the secret sharing scheme. The Eval step implies that it is possible to separately evaluate the function shares, and get the shares of the computed function.

One view of MPC schemes

The "flow" of MPC

Embedded is an idea of a changing distributed data structure, containing a triple $\langle \mathcal{F}, i, f \rangle$, where $\mathcal{F}$ is the function, i is the input and f is the output (eventually the required multi-party computed function value). In the following, $\ldots$ indicates that the value does not matter, and we see how the triple progresses through the protocol:

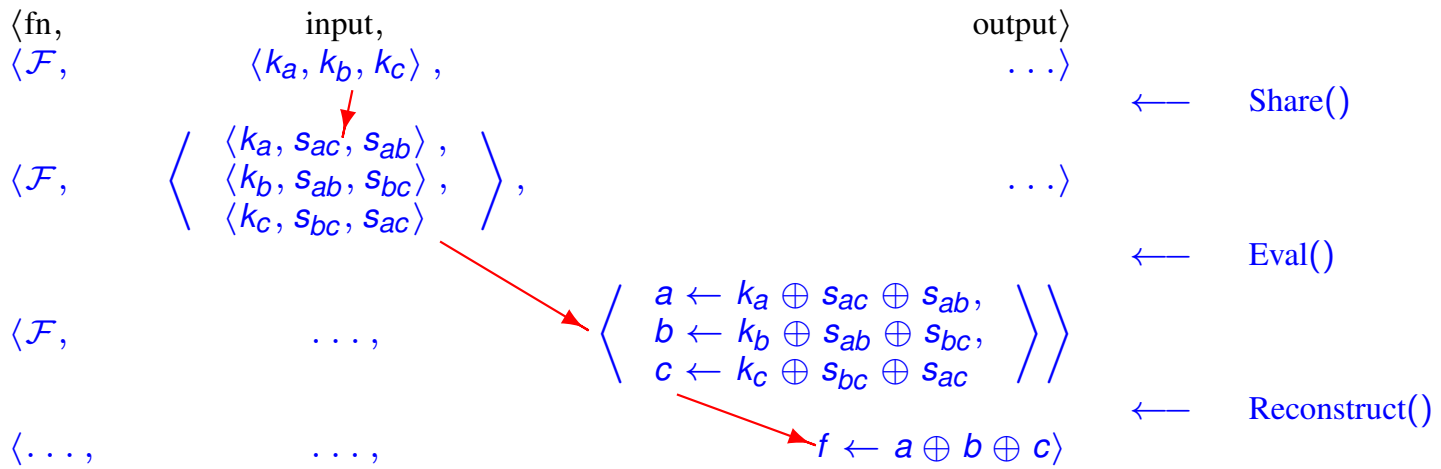


Note that in this view we do not have any encoding or idea of the secrecy requirement for MPC.

Dining cryptographers is an MPC!

The "flow" for dining cryptographers:

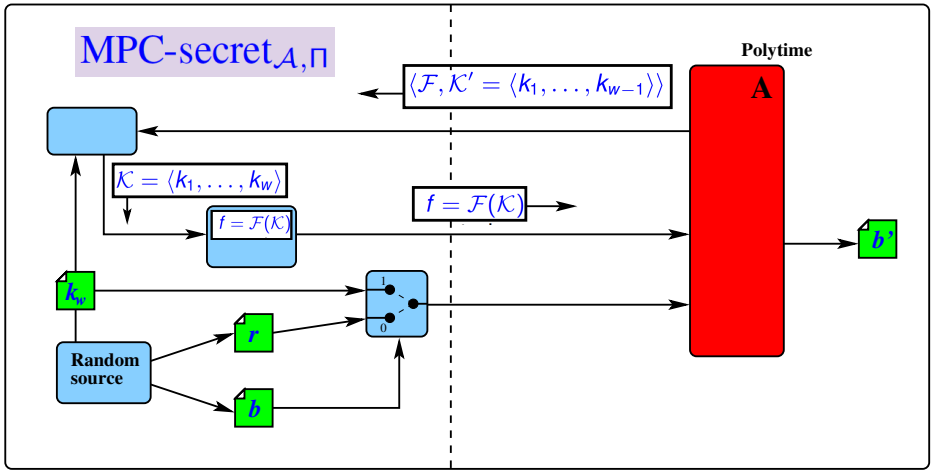
Assume that the cryptographers (Alice, Bob and Charles) each have their private secrets $\langle k_a, k_b, k_c \rangle$, each being a value $(0/1)$ indicating if they paid for the meal. There are two cases - either all are 0, or one is 1. We want to compute the function $f = \mathcal{F}(k_a, k_b, k_c) \stackrel{\text{def}}{=} k_a \oplus k_b \oplus k_c$, which will indicate if one of the cryptographers paid for the meal. The *flow* is as follows, where s_{ab} is a $(0/1)$ secret shared between Alice and Bob:



The final value computed is the desired value:

$$\begin{aligned} f &= a \oplus b \oplus c \\ &= k_a \oplus s_{ac} \oplus s_{ab} \oplus k_b \oplus s_{ab} \oplus s_{bc} \oplus k_c \oplus s_{bc} \oplus s_{ac} \\ &= k_a \oplus k_b \oplus k_c \qquad = \mathcal{F}(k_a, k_b, k_c) \end{aligned}$$

Property: (computational) MPC secrecy



The game: MPC-secret$_{\mathcal{A},\Pi}$ (cannot learn another's key)

1. Adversary submits $\mathcal{F}$ and $\mathcal{K}' = \langle k_1, \ldots, k_{w-1} \rangle$. Challenger generates a random secret k_w, creating $\mathcal{K} = \langle k_1, \ldots, k_w \rangle$ so that it can return $f = \mathcal{F}(\mathcal{K})$, and also (through a random bit b) either k_w or a randomly generated r. Adversary outputs b'.
2. Adversary wins where $b = b'$ and the output is MPC-secret$_{\mathcal{A},\Pi} = 1$.

Definition: computational MPC secrecy

An MPC scheme Π is MPC-secret iff for any $\mathcal{A}, \mathcal{F}$ there is a negl, s.t.

$$\Pr[\text{MPC-secret}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$$

MPC for $+, \times$ using Shamir-style shares

Core ideas for (Shamir-style) addition in MPC

- $(s_{1,j} + s_{2,j}) \bmod p$ is the share for the secret $(k_1 + k_2) \bmod p$ in P_j.

Note that there is no communication among the P_i to evaluate the new shares. So, after the new share is evaluated, each P_i does not gain additional information about the secrets.

Core idea for (Shamir-style) multiplication in MPC

- $(s_{1,j} \times s_{2,j}) \bmod p$ is the share for $(k_1 \times k_2) \bmod p$. But...

Consider what this means. Suppose the $(t-1)$ degree polynomials for k_1, k_2 are $g(x)$ and $f(x)$ respectively, i.e. $g(0) = k_1, f(0) = k_2$ and each P_i has $s_{1,i} = g(i), s_{2,i} = f(i)$. Each P_i can compute $h(i) = (g(i) \times f(i)) \bmod p$, and $h(0) = (k_1 \times k_2) \bmod p$, but now $h(x)$ is a degree $(2t-2)$ polynomial!

We need more parties in MPC to reconstruct for $h(0)$, leading to the condition that $w \geq 2t - 1$ for the (t, w)-threshold scheme.

(Shamir-style) Dimension reduction

To stop needing more parties each time we do a MUL

We need $(2t-1)$ shares to reconstruct $h(0)$, which is $(k_1 \times k_2) \bmod p$. To stop this continual growth, we use dimension reduction. Consider the two functions below, where clearly $h(0) = r(0)$:

$$\begin{aligned} h(x) &= a_0 + a_1 x + \ldots + a_{t-1} x^{t-1} + \ldots + a_{2t-1} x^{2t-1} \\ r(x) &= a_0 + a_1 x + \ldots + a_{t-1} x^{t-1} \end{aligned}$$

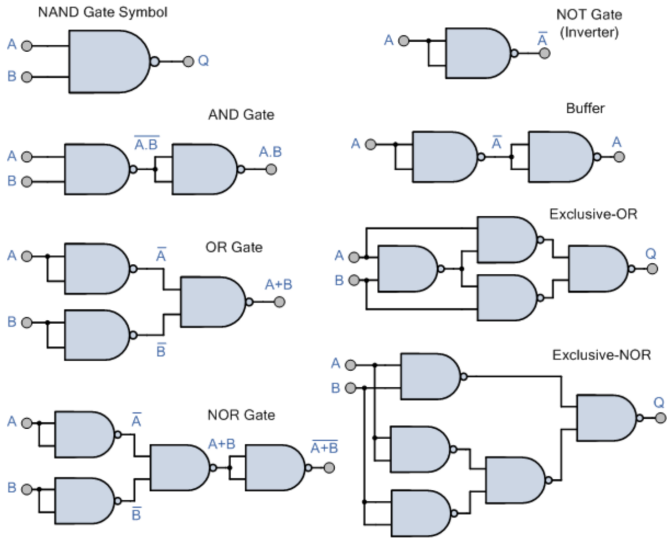
The parties must engage in a subsidiary sharing scheme to obtain the shares of $(k_1 \times k_2) \bmod p$ from the shares of k_1 and k_2 with the following steps:

1. Each P_i computes $h(i)$, treats it as a secret, and (acting as a dealer) distributes the shares of $h(i)$ to everyone.
2. Each P_i, after receiving the shares, computes its share for $r(i)$ for all j.

There are security issues with this as the coefficients of $h(x)$ are directly related to those of $f(x)$ and $g(x)$. For the secure version see `https://dl.acm.org/doi/10.1145/62212.62213`.

MPC for any function/circuit

Symbol	Truth Table		
 2-input NAND Gate	B	A	Q
	0	0	1
	0	1	1
	1	0	1
	1	1	0
Boolean Expression **Q = $\overline{\text{A.B}}$**	Read as A **AND** B gives **NOT** Q		



Computable functions from circuits?

Using MPC for the simple functions ADD and MUL, we can simulate gates. For example, $\text{ADD}_2(k_1, k_2)$ is identical to the XOR gate, $\text{MUL}_2(k_1, k_2)$ is identical to the AND gate, and

$$\text{NAND}_2(k_1, k_2) = \text{ADD}_2(\text{MUL}_2(k_1, k_2), 1)$$

The NAND gate is particularly interesting, as it can be used to construct any other circuit - it is called a *universal* logic gate.

MPC: Secure multi-party computation protocol

Putting it all together

Our goal is for parties to jointly compute a function $y = \mathcal{F}(k_1, k_2, \ldots, k_n)$. Can we design a protocol, where each P_i knows its own secrets, but learns nothing else about other secrets?

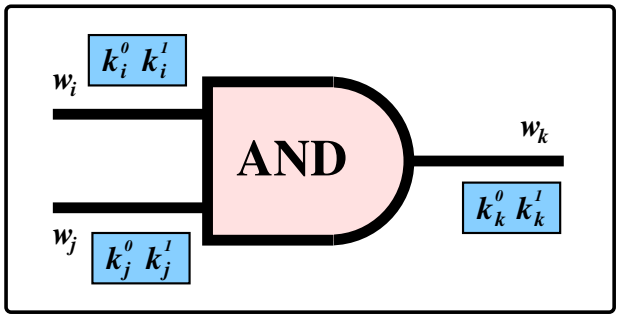
The protocol:

1. Convert the function to an acyclic circuit.
2. Using secret sharing, each P_i distributes k_i as shares to the others.
3. Computations on the shares compute the shares of result y.
4. These shares are combined to get y.

Due to the security of multiplication, to remain secure, the number of colluders must be at most $\frac{1}{3}n$, and because of this, when $n = 2$, the method is not secure. Nevertheless, there is another protocol that supports secure 2-party computation for any function $\mathcal{F}$.

Yao's "Garbled" (one-time) circuits for MPC

w_i	w_j	w_k
0	0	0
0	1	0
1	0	0
1	1	1



Encode the circuits for use in a more efficient way..

Andrew Yao (MIT, Stanford, Berkeley, Tsinghua University, Turing award) introduced labelling of the wires (w_i, w_j, w_k) that interconnect gates of a circuit. Participants can evaluate the output without knowing other inputs. With input labels, decrypt the garbled w_k table to get an output label:

Construct AND label table

w_i	w_j	w_k
k_i^0	k_j^0	$\mathrm{Enc}_{k_i^0}(\mathrm{Enc}_{k_j^0}(\mathrm{Lbl} + k_k^0))$
k_i^0	k_j^1	$\mathrm{Enc}_{k_i^0}(\mathrm{Enc}_{k_j^1}(\mathrm{Lbl} + k_k^0))$
k_i^1	k_j^0	$\mathrm{Enc}_{k_i^1}(\mathrm{Enc}_{k_j^0}(\mathrm{Lbl} + k_k^0))$
k_i^1	k_j^1	$\mathrm{Enc}_{k_i^1}(\mathrm{Enc}_{k_j^1}(\mathrm{Lbl} + k_k^1))$

$\rightarrow$

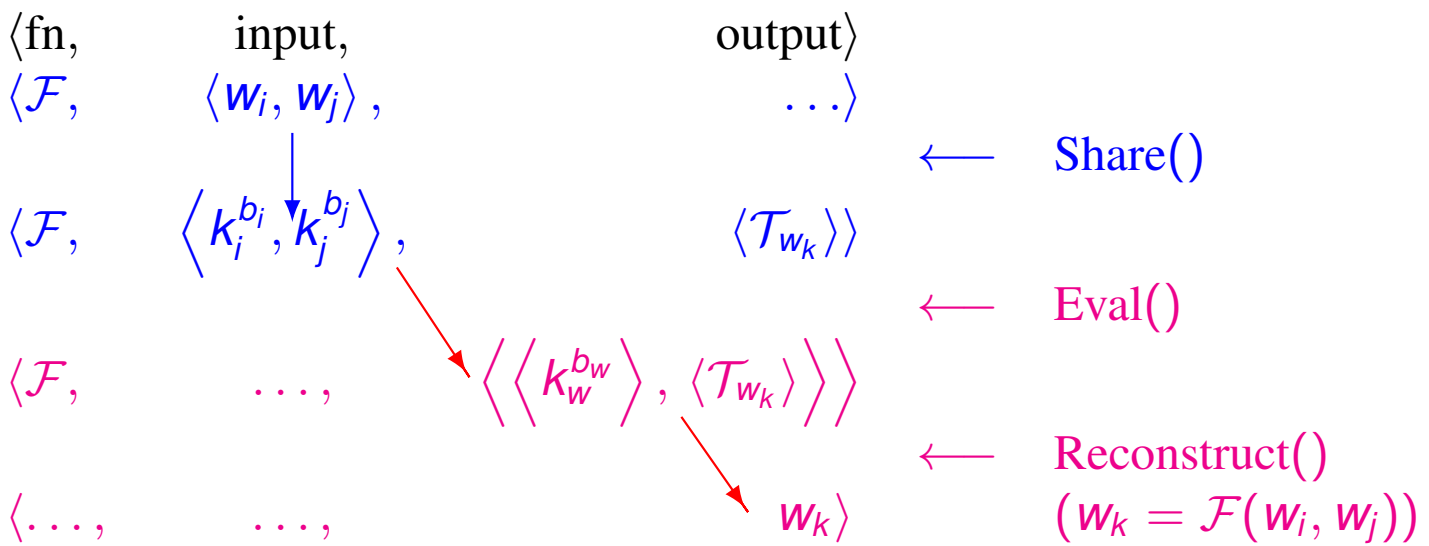
Use Garbled w_k table

w_k
$\mathrm{Enc}_{k_i^0}(\mathrm{Enc}_{k_j^1}(\mathrm{Lbl} + k_k^0))$
$\mathrm{Enc}_{k_i^1}(\mathrm{Enc}_{k_j^1}(\mathrm{Lbl} + k_k^1))$
$\mathrm{Enc}_{k_i^0}(\mathrm{Enc}_{k_j^0}(\mathrm{Lbl} + k_k^0))$
$\mathrm{Enc}_{k_i^1}(\mathrm{Enc}_{k_j^0}(\mathrm{Lbl} + k_k^0))$

Our view of MPC schemes

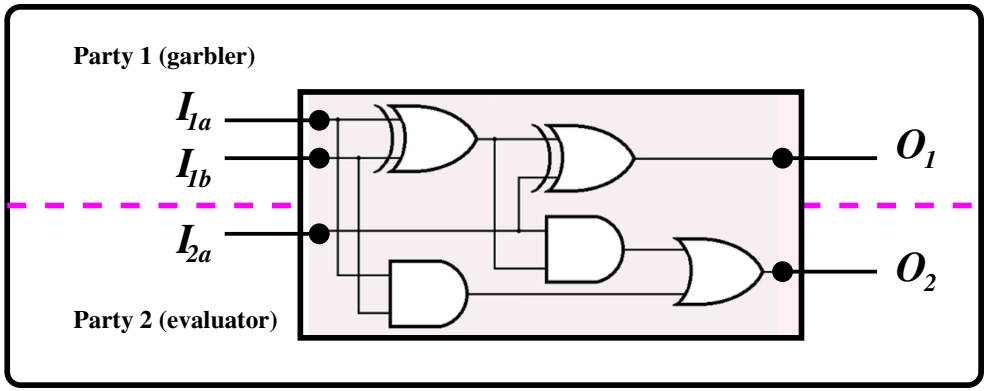
The "flow" of (single) gate evaluation

As before, one of the participants, the garbler, creates (label) tables for the input and output values. The evaluator does not know the meaning of the labels - for example if the evaluator is given k_i^0, it is not aware that it represents 0. The garbler shares the correct label for each input with the evaluator (but not the label/value table). The garbler also outputs the label/value table for the output wire $\mathcal{T}_{w_k} = \begin{cases} k_w^0 & \rightarrow & 0 \\ k_w^1 & \rightarrow & 1 \end{cases}$ and the *flow* is:



In this particular (tiny) example, if $w_k = 1$, the evaluator can infer the input values afterwards, but normally our circuits are much larger.

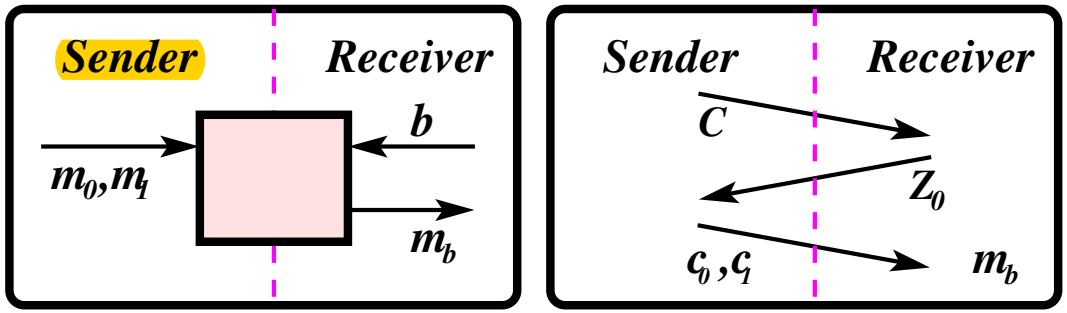
Garbled circuit protocol for (larger) MPC



Using Yao's four step (one time use) protocol...

1. Party 1, the garbler, creates a circuit $\langle \mathcal{C}, \langle I_{1a}, I_{1b}, I_{2a} \rangle, \langle O_1, O_2 \rangle \rangle$, in this case with three inputs and two outputs (one for each party), and sends $\left\langle \mathcal{C}, \left\langle k_{1a}^{b_{1a}}, k_{1b}^{b_{1b}} \right\rangle, \langle \mathcal{T}_{O_2} \rangle \right\rangle$ to Party 2, the evaluator. $\mathcal{T}_{O_2}$ is the label/value table for O_2.
2. Party 1 uses an *oblivious transfer* protocol (shown in the next slide) to send the correct input value for $k_{2a}^{b_{2a}}$ to the evaluator (party 2).
3. Party 2, the evaluator, evaluates the circuit (It now has all the input labels and the O_2 output table), and sends the output label computed for O_1 back to party 1.
4. Party 1 can now find its own output value O_1, as it knows (and in fact created) $\mathcal{T}_{O_1}$.

Construction #4: oblivious transfer (hashes+EC)



Parties agree on an EC group and generator $\langle \mathcal{G}, g \rangle$:

1. The sender sends a uniformly selected point on the curve: $C \in \langle g \rangle$.
2. After selecting n_A, the receiver computes two public keys for its bit b:

$$\begin{aligned} p_k^b &\leftarrow \langle \mathcal{G}, g, \mathcal{X} \rangle = \langle \mathcal{G}, g, n_A g \rangle \\ p_k^{1-b} &\leftarrow \langle \mathcal{G}, g, \mathcal{Y} \rangle = \langle \mathcal{G}, g, C - n_A g \rangle \end{aligned}$$

   and then returns $\mathcal{Z}_0$ (either $\mathcal{X}$ or $\mathcal{Y}$) from $p_k^0 = \langle \mathcal{G}, g, \mathcal{Z}_0 \rangle$ to the sender. Note that p_k^b corresponds to a secret key $s_k = \langle \mathcal{G}, g, n_A \rangle$.
3. The sender computes the other key using $\mathcal{Z}_1 = C - \mathcal{Z}_0$, not knowing which is which, before uniformly choosing k, and encrypting both messages with $\mathcal{H} : \mathcal{G} \to \mathbb{Z}_n$: $c_0 = \langle m_0 \oplus \mathcal{H}(k\mathcal{Z}_0), kg \rangle$ and $c_1 = \langle m_1 \oplus \mathcal{H}(k\mathcal{Z}_1), kg \rangle$.

Only the message corresponding to b will decrypt correctly!

A new construction for ENC_{p_k} (hashes+EC)

GenECC: creates public keys p_k and s_k

1. Using an (additive) elliptic group $\mathcal{G} = E_p(a, b)$, select a generator g in the group which has a large order n.
2. Uniformly choose $n_A : n_A < n$. s_k is $\langle \mathcal{G}, g, n_A \rangle$
3. Calculate $P_A = n_A g$. p_k is $\langle \mathcal{G}, g, P_A \rangle$

Construction #5: ENC_{p_k} for bit message m

$\mathrm{Gen}(1^n)$: Compute $(p_k, s_k) \leftarrow \mathrm{GenECC}(1^n)$.

$\mathrm{Enc}_{p_k}(m)$: Given p_k, message $m \in \mathbb{Z}_n$, k uniform $k \in \mathbb{Z}_n^*$, and $\mathcal{H} : \mathcal{G} \to \mathbb{Z}_n$, encrypt like this: $\langle c_1, c_2 \rangle \leftarrow \langle m \oplus \mathcal{H}(kP_A), kg \rangle$.

$\mathrm{Dec}_{s_k}(\langle c_1, c_2 \rangle)$: Given $s_k, \langle c_1, c_2 \rangle$, calculate: $m \leftarrow c_1 \oplus \mathcal{H}(n_A c_2)$.

Correctness

$$c_1 \oplus \mathcal{H}(n_A c_2) = m \oplus \mathcal{H}(kP_A) \oplus \mathcal{H}(n_A kg) = m \oplus \mathcal{H}(kn_A g) \oplus \mathcal{H}(n_A kg) = m$$

Wrapping up: in this course, and the future...

Applications: it is not just encryption!

Formal (mathematical) crypto has found its way into a range of application areas: **public-key encryption, signatures, IBE, MPC, MPKE,** non-interactive zero knowledge (NIZK) and non-interactive witness-indistinguishable (NIWI) proof systems, time-locked encryption, witness encryptions, 2-round MPC, attribute-based encryption, deniable encryption, functional encryption, quantum money...

Building blocks: It is not just factorizing and Dlog!

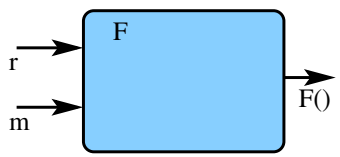
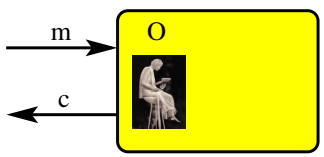
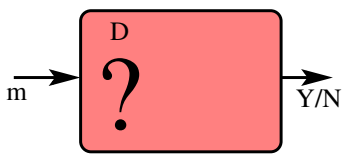
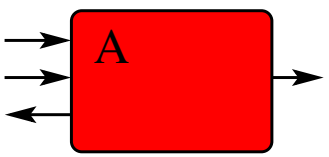
In a sense, the core of modern crypto centres on complexity ideas - hard vs easy, $\mathrm{P} \neq \mathrm{NP}$, and though the notions of the (hopefully hard) factorization and Dlog based constructions have worked so far, we already know of polynomial-time (quantum) solutions for most of the systems based on these. This drives the interest in other bases, perhaps ones with relevance in the post-quantum world.

A not-complete list of core hardness ideas that have been used, or could be applied: **factorization, Dlog, bilinear maps**, learning with errors (LWE), learning parity with noise (LPN), multilinear maps, iO ideas, latticed-based crypto...

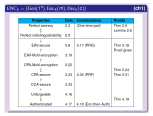
CS4236 concerns

A compact view...

Jargon/Examples	**Proof**	**Representations**	**Data**
Adversary,CPA	Detailed, skeletons	1^n, $\{0,1\}^n$	0110101110...
Attacks	...suppose not ...	$a \times b \rightarrow c$	
Systems	Proof by construction		

Functions	**Oracles**	**Distinguishers**	**Adversaries**
			

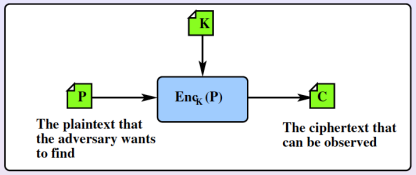
Constructions	**Game/Experiments**	**System/Schemes Π**	**Properties**
ECC, IBE, RSA	$\text{Name}^{\text{context}}_{\mathcal{A},\Pi}(\text{param})$	(Enc, Dec, Gen)	CCA-secure
MPC, OAEP	($\mathcal{A}$ is adversary)	(Gen, Mac, Vrfy)	Unforgeable

Randomness	**Probability/Negligable**	**Complexity**	**Math Ecosystem**
true: r, $f(\cdot)$	Bayes/distributions	$\mathcal{O}(n)$	
PRG: $G()$	...there is a negl s.t.	PPT, exp	

Math building blocks 1

Probability theory

Probability theory $\Pr[E_1|E_2] = \frac{\Pr[E_2|E_1]\times\Pr[E_1]}{\Pr[E_2]}$



Relevance to crypto

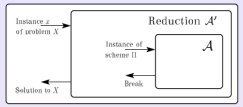
In the context of cryptography, one can imagine throwing two independent dice. One of them is the message P, and the other the key K. Next, a ciphertext is computed from the plaintext and key. Since the key and plaintext are randomly chosen, the ciphertext C is thus also random.

Now, an adversary has observed the ciphertext and thus knows that C=3. The adversary wants to know what are the chances that the plaintext is 5, given that the ciphertext is 3, that is, the probability

$$\Pr[P = 5|C = 3]$$

Proof styles

Proof styles



Outline of reduction proofs (p65)

Proof that some new construction Π is secure if some underlying cryptographic primitive X is secure. We often have this situation - for example Diffie-Hellman and factoring large numbers. We start with "Suppose not".

Assume that X is secure but Π is insecure, so this means that a PPT adversary $\mathcal{A}$ attacks Π with success probability $\epsilon(n)$, not $\mathrm{negl}(n)$.

- Make a PPT algorithm $\mathcal{A}'$ that tries to solve instance x of X using $\mathcal{A}$.
- If $\mathcal{A}$ succeeds in breaking Π, $\mathcal{A}'$ solves x with probability $\frac{1}{p(n)}$.
- $\mathcal{A}$ and $\mathcal{A}'$ solve X with probability $\frac{\epsilon(n)}{p(n)}$. $\mathcal{A}$ and $\mathcal{A}'$ are both PPT.

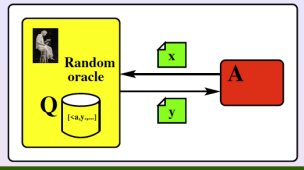
X is solved by PPT $\mathcal{A}'$ with non-negligable probability. A contradiction, so

$$X \text{ is secure} \longrightarrow \Pi \text{ is secure}$$

■

The random oracle

The random oracle (for hashes)



An idealised model, for use in proofs

In the definition of the PRF before, we saw similar oracles. The "random oracle" is an alternative way, a methodology, for formulating hash ideas. It makes proofs (sometimes) a little simpler. However, a random oracle is not a realizable, actual thing!

It is an idealized model, and so any proofs based on an assumption that (say) the underlying hash is a random oracle are theoretically meaningless. We really should not be saying

...xxx is secure, assuming $\mathcal{H}$ is a random oracle.

Statements that are made without assuming a random oracle are called "*Standard model*". (e.g. xxx is secure under standard model).

Euler and Fermat

Euler and Fermat

$\mathbb{Z}_p$ $\mathbb{Z}_N$

Euler's theorem: (composites)

If N is any positive integer and a is any positive integer less than N with no divisors in common with N, then

$$a^{\phi(N)} \bmod N = 1$$

where $\phi(N)$ is the *Euler phi (totient) function*:

$$\phi(N) = N(1 - 1/p_1)\ldots(1 - 1/p_m)$$

and $p_1, \ldots, p_m$ are all the prime numbers that divide evenly into N.

If N is a prime, then using the formula, we have

$$\phi(N) = N(1 - \frac{1}{N}) = N(\frac{N-1}{N}) = N - 1$$

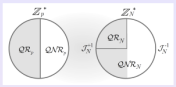
We see that Fermat's result is a special case of Euler's:

$$a^{\phi(N)} \bmod N = a^{N-1} \bmod N = 1$$

Math building blocks 2

Legendre and Jacobi

Identifying quadratic residues


The Legendre and Jacobi "symbol" notation

Legendre: *identifies* the QR in $\mathbb{Z}_p$; a function of a and p: $(\mathbb{Z}_p)$

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \mod p = \begin{cases} 1 & \text{if } a \text{ is a QR} \\ -1 & \text{if } a \text{ is not a QR} \\ 0 & \text{if } a = 0 \mod p \end{cases}$$

Jacobi: is a generalization of Legendre, extending it to $\mathbb{Z}_N$ (composites). $(\mathbb{Z}_N)$ Easy to compute given it's prime factorization. For example, given $N = p \times q$ with $p \neq q$, then

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

Note that there is an algorithm that, given a and N, can find the Jacobi symbol without knowing the factorization of N. From the Jacobi symbol, we can't tell whether a is a QR, but no entry with -1 is a quadratic residue.

Hard and easy

Hard and easy (Root finding, Dlog,...)

The difficult problem in RSA: find e^{th} root in $\mathbb{Z}_N^*$

Given N, e, y, find x s.t. $x^e \mod N = y$.

Root finding can occur in any group, and there are efficient algorithms when N is prime. The hardness of RSA requires N to be a composite and thus relates to factorization, which only makes sense in a ring with two operators $\times, +$. It is not easy to generalize RSA encryption to other algebraic groups or rings.

The difficult problem in other systems: solve discrete log

Given N, x, y, find e s.t. $x^e \mod N = y$.

Such problems also naturally occur in any group. It turns out that there many groups whereby these problems seem to be very difficult, and at the same time, with certain useful properties (e.g. bilinear maps). This flexibility makes it attractive to design crypto systems based on DL, DDH or CDH. (there are many variants, e.g. knowledge of exponent, generalized DDH, etc, and many choices of groups, e.g Elliptic Curve, bilinear map, etc).

Cyclic groups

Cyclic groups

a	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}
2	2	4	8	5	10	9	7	3	6	1
3	3	9	5	4	1	3	9	5	4	1
4	4	5	9	3	1	4	5	9	3	1
5	5	3	4	9	1	5	3	4	9	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1
9	9	4	3	5	1	9	4	3	5	1
10	10	1	10	1	10	1	10	1	10	1

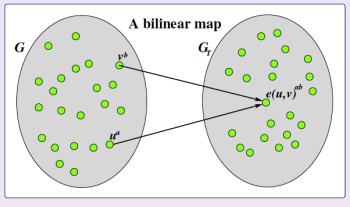
Cyclic groups and subgroups

All elements in $\mathbb{Z}_{11}^*$ can be obtained by successive multiplication by 6. So this is a cyclic group and can be written as $\langle 6 \rangle$. 6 is called a generator.

In general, when p is a prime, for any g in $\mathbb{Z}_p^*$, $\langle g \rangle$ forms a cyclic group or subgroup of $\mathbb{Z}_p^*$, . The order of this group is the number of elements in it, and so for $\langle 6 \rangle = \mathbb{Z}_{11}^*$, we have the order $|\langle 6 \rangle| = 10$. By contrast, $\langle 3 \rangle$ forms a cyclic subgroup of $\mathbb{Z}_{11}^*$ of order 5.

Bilinear maps

Bilinear maps (or pairings)



Pairings ...

A (symmetric style) bilinear map is a pairing between two elements of a group to a second (same order) group: $G \times G \to G_T$. There are also (asymmetric) pairings like $G_1 \times G_2 \to G_T$.

Such a symmetric mapping (if it can be efficiently computed) can be used to solve DDH: $z = ab$ iff $e(g, g^z) = e(g^a, g^b)$. Originally this was the focus of bilinear maps in crypto, and this is why CDH is considered *harder* than DDH.

The other thing is that it can allow you to reduce the discrete log problem in G to a (perhaps simpler) discrete log problem in G_T: $e(g, g^a) = e(g, g)^a$.

The math "ecosystem" in this course

Math ecosystem/framework elements:

- **Systems/schemes** which are the basic elements of the crypto world.
- **Security properties** over schemes defined by **games/experiments**.
- **Constructions**, which are instances of schemes, and
- **Proofs**, to show relations over properties and constructions.

The 11 schemes (with properties) are...

ENC_k: Symmetric encryption - EAV, CPA, CCA, unforgeability

MAC_k: Existential unforgeability, strongly-secure

HASH: Collision resistant

COM: Commitment - Binding+hiding (together secure)

ENC_{p_k}: Asymmetric encryption - DDH, Factor, RSA, EAV, CPA, CCA

IBE_{p_k}: Boneh/Franklin (construction only)

SIG_{p_k}: Existential unforgeability

$\text{KA}, \text{KEM}_{p_k}$: SecKA, DHKA, KE-eav, CPA, CCA

SS: Collusion resistant Shamir, Feldman

MPC: MPC-secret Dining cryptographers, $+$, $\times$ garbled circuits

Symmetric encryption schemes

$$\mathrm{ENC}_k = (\mathrm{Gen}(1^n), \mathrm{Enc}_k(m), \mathrm{Dec}_k(c)) \qquad \textbf{(ch1)}$$

Properties	Defs	Constructions	Proofs
Perfect secrecy	2.3	(One time pad)	Thm 2.9
↕			Lemma 2.6
Perfect indistinguishability	2.5		
↓			
EAV-secure	3.8	3.17 (PRG)	Thm 3.18
↑			Proof given
EAV-Multi-encryption	3.19		
↑			
CPA-Multi-encryption	3.22		
↕			Thm 3.24
CPA-secure	3.23	3.30 (PRF)	Thm 3.31
↑			
CCA-secure	3.33		
+			
Unforgeable	4.16		
↕			Thm 4.19
Authenticated	4.17	4.18 (Enc-then-Auth)	

MAC and HASH schemes

$$\mathrm{MAC}_k = (\mathrm{Gen}(1^n), \mathrm{Mac}_k(m), \mathrm{Vrfy}_k(m,t)) \quad \textbf{(4.1)}$$

Properties	Defs	Constructions	Proofs
Unforgeable	4.2		
↑			Discussed
Strong	4.3	4.5 (Using PRF)	Thm 4.5
		4.7 (Variable length)	Thm 4.8
		4.11 (CBC-MAC)	Thm 4.12
		4.11(a) (Variable CBC-MAC)	

$$\mathrm{HASH} = (\mathrm{Gen}(1^n), \mathcal{H}^s(x)) \quad \textbf{(5.1)}$$

Properties	Defs	Constructions	Proofs
CollisionResistant	5.2	5.3 (Long message)	

Asymmetric encryption schemes

$$\mathrm{ENC}_{p_k} = (\mathrm{Gen}(1^n), \mathrm{Enc}_{p_k}(m), \mathrm{Dec}_{s_k}(c)) \quad \textbf{(11.1)}$$

Properties	Defs	Constructions	Proofs
		11.26 (Textbook RSA) (Elgamal) (ECC)	Thm 11.18
DDH-hard Factor-hard ↑ RSA-hard	8.63 8.45 8.46		
EAV-secure ↕ CPA-secure ↑ CCA-secure	11.2	11.32 (1-Bit RSA)	

Commitment, KEM, IBE schemes

$$\mathrm{COM} = (\mathrm{Setup}(1^n), \mathrm{Commit}(a), \mathrm{Open}(c_a)) \quad \textbf{(ch5)}$$

Properties	Defs	Constructions	Proofs
Secure (Binding+Hiding)	5.13	(Hash based system)	

$$\mathrm{KEM}_{p_k} = (\mathrm{Gen}(1^n), \mathrm{Encaps}_{p_k}(1^n), \mathrm{Decaps}_{s_k}(c)) \quad \textbf{(11.9)}$$

Properties	Defs	Constructions	Proofs
CPA-secure		11.10 (CPA_KEM+EAV_ENC)	Thm 11.12
CCA-secure	11.13	11.10 (CCA_KEM+CCA_ENC)	Thm 11.13
		11.37 (CCA_KEM+TextbookRSA)	

$$\mathrm{IBE}_{p_k} = (\mathrm{Setup}(1^n), \mathrm{Enc}_{\mathrm{nm},M_P}(m), \mathrm{Dec}_{k_U,M_P}(c)) \quad \textbf{(...)}$$

Properties	Defs	Constructions	Proofs
		(Boneh/Franklin)	

KA and Signature schemes

$$\mathrm{KA} = (\mathrm{Init}(1^n, \overline{p}), \mathrm{DoProt}(\overline{p})) \qquad \textbf{(...)}$$

Properties	Defs	Constructions	Proofs
SecKA	10.1	(DHKE) 3-party Bilinear maps	Thm 10.3

$$\mathrm{SIG}_{p_k} = (\mathrm{Gen}(1^n), \mathrm{Sign}_{s_k}(m), \mathrm{Vrfy}_{v_k}(m, t)) \qquad \textbf{(12.1)}$$

Properties	Defs	Constructions	Proofs
Unforgeable	12.2	12.3 (Hash and Sign) 12.36 (Hash and Sign, RSA) (Elgamal Sign)	Thm 12.4

Secret sharing, and MPC schemes

$$\mathrm{SS} = (\mathrm{Init}(n, w), \mathrm{Share}(k, t, \mathcal{P}), \mathrm{Reconstruct}(S)) \quad \textbf{(...)}$$

Properties Defs	Constructions	Proofs
Collusion resistant	(Threshold SS) (Modulo attempt) (Shamir's (t, w)) (Feldman's VSS)	

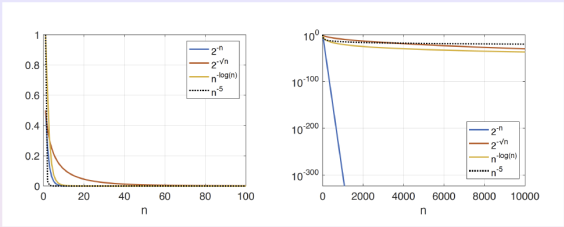
$$\mathrm{MPC} = (\mathrm{Init}(n), \mathrm{Share}(k), \mathrm{Eval}(\mathcal{S}), \mathrm{Reconstruct}(\mathcal{F}_s)) \quad \textbf{(...)}$$

Properties Defs	Constructions	Proofs
MPC-secret	(Dining cryptographers) (Addition, Multiplication) (Yao's garbled circuits)	

Functions and structures we saw

Negligable functions

Negligable functions



Definition 3.4 - negligable function definition

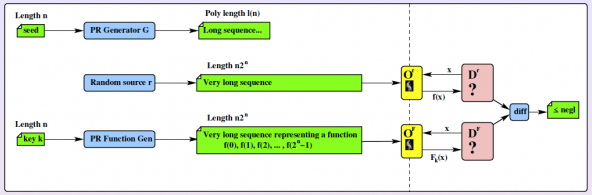
A non-negative function f from natural to real numbers $f : [0, 1 \ldots] \to [0, \infty)$, is *negligible* if there is an N s.t. for all $n > N, f(n) < \frac{1}{p(n)}$. It is asymptotically smaller than any inverse polynomial function.

Properties of negl

1. Proposition 3.6. $\text{negl}_1 + \text{negl}_2$ is *negligible*
2. $p(n) \times \text{negl}$, is *negligible* where $p(n)$ is any polynomial.

PRGs and PRFs

Pseudo-random generators and functions



Definition 3.25

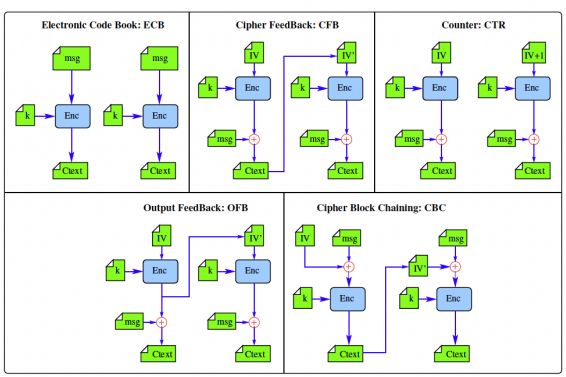
Let $F : \{0,1\}^* \times \{0,1\}^* \longrightarrow \{0,1\}^*$ be an efficient length-preserving keyed function. F is a pseudorandom function if for all PPT distinguishers D, there is a negl s.t.

$$\left|\Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]\right| \leq \text{negl}(n)$$

The first probability is taken over choice of the key k, randomness of D. The second probability is taken over choice of f from Func_n .
(Here, the size of the key is the same as the message. In general, length-preserving only refers to the size of message and ciphertext.)

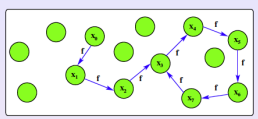
Modes

Modes



Floyd

Floyd - for chains like $x_0 \to f(x_0) \to f(f(x_0)) \to \ldots$



Find loop with $f(x) \stackrel{\text{def}}{=} \mathcal{H}(x)$; time $\mathcal{O}(2^n)$, and memory $\mathcal{O}(1)$

(Robert W.) Floyd's cycle-detection algorithm: the tortoise t and the hare h. Variables hop through a graph at different rates, stopping when they collide.

Algorithm 5.9: part 1 finds the loop in a (digest/hash) chain

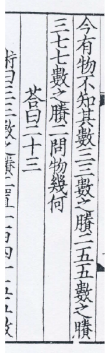
```
def H(x):
    return md5(x.encode('utf-8')).hexdigest()
def f(x):
    return H(x)[:10]

def floyd(x0):
    t = f(x0)
    h = f(f(x0))
    print('Find the loop')
    while (t != h):
        t = f(t)
        h = f(f(h))
```

Algorithms we saw

CRT

CRT (Chinese Remainder Theorem)



Problem posed by Wei/Jin mathematician Sun Tzu 1700 years ago:

There are certain things whose number is unknown. Repeatedly divided by 3, the remainder is 2; by 5 the remainder is 3; and by 7 the remainder is 2. What will be the number?

The Chinese remainder theorem - nowadays

Two simultaneous congruences $n = n_1 \bmod m_1$ and $n = n_2 \bmod m_2$ are only solvable when $n_1 = n_2 \bmod (\gcd(m_1, m_2))$. The solution is unique modulo $\mathrm{lcm}(m_1, m_2)$.

It demonstrates to us that a number less than the product of two primes can be uniquely identified by its residue modulo those primes. It is useful in RSA.

Extended Euclidean

Extended Euclidean algorithm

An interesting property

If the $\gcd(a, b) = r$ then there exist integers m and n so that $ma + nb = r$. Use to calculate the multiplicative inverse of an element x modulo n.

The extended Euclidean algorithm

1. We begin by dividing n by x, and as we carry out each step i of the Euclidean algorithm discovering the quotient q_i, we also calculate an extra number x_i. For the first two steps $x_0 = 0$ and $x_1 = 1$.
2. For the following steps, calculate $x_i = x_{i-2} - x_{i-1} q_{i-2} \bmod n$.
3. If the last non-zero remainder occurs at step k, then if this remainder is 1, x has an inverse and it is x_{k+2}.

The inverse of 15 modulo 26, showing the method:

$$
\begin{array}{lrcl}
0: & 26 & = & 1 * 15 + 11 \\
1: & 15 & = & 1 * 11 + 4 \\
2: & 11 & = & 2 * 4 + 3 \\
3: & 4 & = & 1 * 3 + 1 \\
4: & 3 & = & 3 * 1 + 0
\end{array}
\qquad
\begin{array}{lcl}
x_0 = 0 & & \\
x_1 = 1 & & \\
x_2 = 0 - 1 * 1 \bmod 26 & = & 25 \\
x_3 = 1 - 25 * 1 \bmod 26 & = & 2 \\
x_4 = 25 - 2 * 2 \bmod 26 & = & 21 \\
x_5 = 2 - 21 * 1 \bmod 26 & = & 7
\end{array}
$$

The bigger crypto constructions

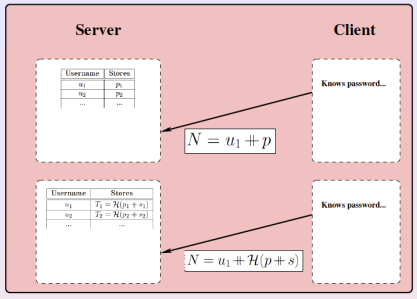
The constructions were...

Enc-Auth:	Enc-then-Auth - authentic and CCA secure
MD/HMAC:	Merkle/Damgård and HMAC for variable lengths
RSA:	Encryption, CCA-secure KEM
OAEP:	Feistel based padding scheme
DHKE:	Diffie/Hellman key exchange
Elgamal:	Encryption, signatures
IBE:	Boneh/Franklin IBE based on bilinear maps
ECC:	Encryption, ECC/DHKE
Hash&Sign:	For secure signatures
MPC:	Multi-party computations

Applications (apart from the obvious)

Passwords and hashing

Passwords and hashing



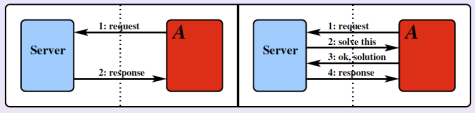
Encryption $\neq$ Hashing... Basic authentication is not much

Assume $\mathcal{H}$ is a cryptographic hash function, p is the password, s is a salt for the password, and $p + s$ as before.

Note that here, a hash is as good as the password.

Proof of work

Proof-of-work



NDSS 1999

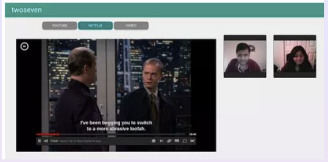
In a typical Denial of Service attack, the attacker invests relatively less resource in issuing a request, but the victim has to spend much more resources in answering the request. The puzzle based method tries to flip this asymmetry. An attacker needs to submit a proof-of-work before the server serves the request.

The puzzles should be easy to construct, easy to verify, but difficult to solve. One way to construct a puzzle might be to choose a random string s, and force the attacker to find a string r s.t. the first (or last) 20 bits of the digest $\mathcal{H}(r + s)$ are all zeros.
The attacker is forced to carried out an expected number of 2^{20} hash operations (?) to find such a choice of r.

https://www.ndss-symposium.org/wp-content/uploads/2017/09/A-Cryptographic-Defense-Against-Connection-Depletion-Attacks-Ari-Juels.pdf

Commitment

Commitment



Movie night

Alice and Bob are talking over a secure channel. They want to decide which movie to watch. Alice prefers *movieA*, but Bob wants *movieB*.

To resolve that, they derive this protocol: each of them is to think of a bit, say a and b. If $(a \oplus b) = 1$, then they will watch *movieA*, otherwise *movieB*.

Now, how to exchange these two bits a and b so that they are unable to cheat? If Alice sends Bob $a = 1$ first, Bob can force Alice to watch *MovieB*.

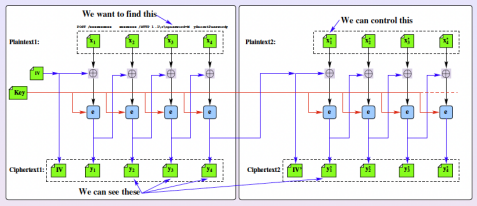
Tossing a coin?

Alice and Bob are not in the same room. It is problematic.

Attacks #1

Rizzo/Duong (CPA)

CPA attack - Rizzo/Duong



BEAST: Browser Exploit Against SSL/TLS

The attack is exploited by Rizzo & Duong on TLS (BEAST).
A malicious site with some javascript/java may convince a browser to send crafted messages (A CPA), and by observing the encrypted messages, perhaps discover an important cookie or token.

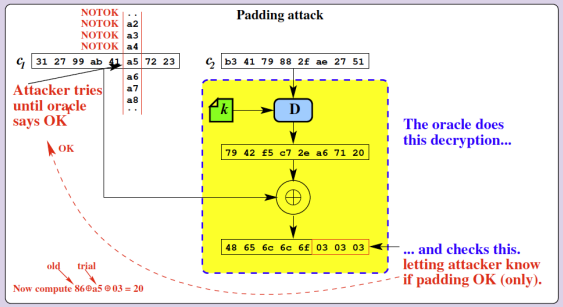
Description

https://www.exploit-db.com/docs/english/15137-practical-padding-oracle-attacks.pdf

Padding oracle (CCA)

CCA attack - padding oracle

Attacker discovers third-to-last byte...

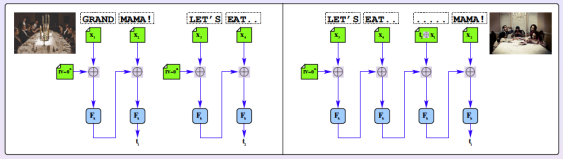


Since we know (now) the last two bytes (02 02), we can set the last two bytes to 03 03.

Now we try all the third-to-last values in c_1 and uncover 20.

Concatenation (CPA)

CPA concatenation attack (on CBC-MAC)



Construction 4.11 - Fixed-length CBC-MAC (p123)

With key $k \in \{0,1\}^n$, PRF F_k, message length $\ell \times n$, $t_0 := 0^n$, then we have:

$\mathrm{Mac}_k(m)$: For $i = 1, ..., \ell$: $t_i := F_k(t_{i-1} \oplus m_i)$. Then output the tag $t = t_\ell$.

$\mathrm{Vrfy}_k(m, t)$: output 1 if and only if $t = \mathrm{Mac}_k(m)$.

Theorem 4.12

If F_k is PRF, and m fixed length, CBC-MAC is secure.

Concatenation attack possible for arbitrary length m

If the adversary has obtained two pairs (m_1, t_1) and (m_2, t_2) - then m_3 can be easily created with a valid tag t_1 or t_2. In the above case (m_3, t_1) passes Vrfy.

Rainbow tables

Rainbow table attack

$$w_0 \xrightarrow{\mathcal{H}} y_0 \xrightarrow{R} w_1 \xrightarrow{\mathcal{H}} y_1 \xrightarrow{R} w_2 \xrightarrow{\mathcal{H}} y_2 \xrightarrow{R} w_3 \xrightarrow{\mathcal{H}} y_3$$

$$w_9 \xrightarrow{\mathcal{H}} y_9 \xrightarrow{R} w_2 \xrightarrow{\mathcal{H}} y_2 \xrightarrow{R} w_3 \xrightarrow{\mathcal{H}} y_3 \xrightarrow{R} w_4 \xrightarrow{\mathcal{H}} y_4$$

Collisions of the reduce function may happen frequently

The pairs $\langle w_0, y_3 \rangle$ and $\langle w_9, y_4 \rangle$ will be stored. This causes two issues:

1. Efficiency in storage: Part of the chain is duplicated. The words w_2 and w_3 appear in the table twice.
2. Efficiency in search: Lead to searches in the wrong chains, before hitting the right chain. For the query y_9, the lookup process would transverse both chains.

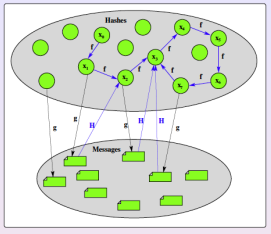
Solution to the problem: the "Rainbow" table

Suppose there are k reduce operations and k hash operations in the chain, instead of using the same function $R()$ in all the reduce operations, the Rainbow table uses k different functions. The first reduce is $R_1()$, followed by $R_2()$ and so on.

Attacks #2

Hash collision

Hash collision attack



Apply Floyd by using $f(x) \stackrel{\text{def}}{=} \mathcal{H}(g(x))$

Consider messages $m \in M$, digests $h \in H$, the functions $g : H \to M$, and $f : H \to H$ defined as $f(x) \stackrel{\text{def}}{=} \mathcal{H}(g(x))$. The elements in the *chain/trail* of digests are mapped to the messages. In the textbook, p168, there is an example with *good*, and *bad* messages about Bob:

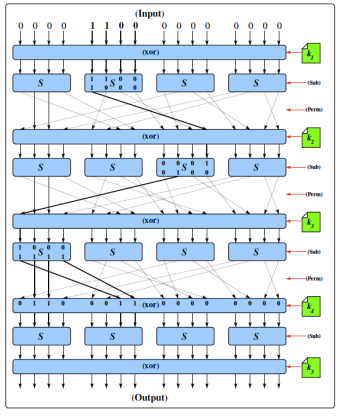
$g(0000)$ = Bob is a good and honest worker.
$g(0001)$ = Bob is a difficult and....

Differential cryptanalysis

Differential cryptanalysis attack

x	$S(x)$
0	0
1	b
2	5
3	1
4	6
5	8
6	d
7	4
8	f
9	7
a	2
b	c
c	9
d	3
e	e
f	a

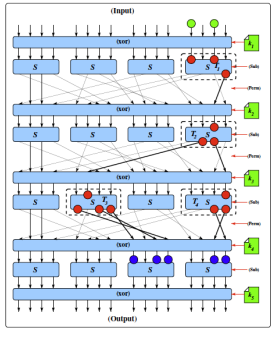
z	$P(z)$
1	7
2	2
3	3
4	8
5	12
6	5
7	11
8	9
9	10
10	1
11	14
12	13
13	4
14	6
15	16
16	15



Linear cryptanalysis

Linear cryptanalysis attack

Note that the bias from the red dots/bits $\varepsilon(T_1 \oplus T_2 \oplus T_3 \oplus T_4)$ is exactly the bias of the green and blue bits.



Linear bias of the whole SPN

To use the piling lemma, we assume that T_1, T_2, T_3, T_4 are independent , but of course they are not. However, it is a good approximation.

Birthday/factorization

Birthday attack (on factorization)

Pollard Rho Algorithm - a birthday attack!

As we saw before, a straightforward factorization algorithm takes $\mathcal{O}(N^{\frac{1}{2}})$ time. By contrast, this algorithm takes an expected $\mathcal{O}(N^{\frac{1}{4}})$ time[a]. Interestingly, it uses a birthday attack, such as we looked at a few weeks ago. Just for curiosity, let's look at the details.

[a] If N is 200 bits, then the security is about 50 bits under this attack.

Pollard Rho Algorithm – improvement to $\mathcal{O}(N^{\frac{1}{4}})$

Given $N = p \times q$, and where $p < q$, an important observation is that for any x, x' s.t. (a) $x \neq x'$ and (b) $x = x' \bmod p$, then $\gcd(x - x', N) = p$. Why?

A birthday attack! If we have a set X of integers (randomly drawn from $\mathbb{Z}_N$) with $|X| \approx 1.177 p^{\frac{1}{2}}$, then by the birthday attack, the chance that there are two elements x, x' s.t. (a) $x \neq x'$ and (b) $x = x' \bmod p$ is more than 50%. Hence, potentially, we have a $\mathcal{O}(p^{\frac{1}{2}}) \approx \mathcal{O}(N^{\frac{1}{4}})$ algorithm to factorize N.

Now we need an algorithm to find two elements in the set X which have the same reminder modulo p. Hmmmmm... sounds like a collision...

Attacks #3

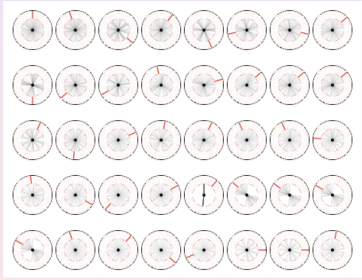
Quantum (Shor)

Quantum computation attack (Shor)

Find repetition values for all powers simultaneously:

A Quantum register holds all possible values at the same time, and we perform an amplifying operation, that only leaves stable repetition values.

Mark the same times every day... In the simulation, only the clock with a repetition rate that we are interested in remains:



Padding attack on RSA (CCA)

CCA padding attack (on RSA)

PKCS#1 padding attack on KEM, from 1998

00	02	padding string	00	data block

In 1998, Daniel Bleichenbacher from Bell labs published a padding oracle attack on PKCS#1 (and hence SSL V3.0). The attack is based on the fact that RSA is homomorphic with respect to multiplication. In addition, some (web) servers can act as padding oracles.

In SSL, during the initial negotiation, critical keys are sent, encrypted using RSA (a KEM). A padding oracle attack can retrieve this "pre"master secret key, in what has been called the “million message” attack. The details of this attack are found in the paper, but it is not dissimilar to the padding oracle attack in Session 5; multiple messages are sent to the server, which responds differently if the format of the padding is correct or incorrect.

“Fixing” PKCS#1

The approach taken to fix this was to ask vendors to add extra countermeasures: primarily returning uninformative server responses.

DDH attack using QR

QR attack (on DDH)

Consider definition 10.1 - eav-security

An adversary wins if it distinguishes between a random key $k' = g^z$ and the real key g^{ab}. It has the transcript `trans`, which in DH contains g^a, g^b, g and p.

This is exactly the DDH problem, distinguishing between $A = (g^a, g^b, g^z)$, and $B = (g^a, g^b, g^{ab})$. In the group $\mathbb{Z}_p^*$, DDH is easy to decide using the Legendre symbol. If a value such as g^a has a QR, then a must be even. We can work out the Legendre symbol for each of A or B above. For A, $\left(\frac{g^z}{p}\right)$ will be random, whereas for B the only possibilities are as follows:

$\left(\frac{g^a}{p}\right)$		$\left(\frac{g^b}{p}\right)$		$\left(\frac{g^{ab}}{p}\right)$	
1	(a even)	1	(b even)	1	(ab even)
1	(a even)	−1	(b odd)	1	(ab even)
−1	(a odd)	1	(b even)	1	(ab even)
−1	(a odd)	−1	(b odd)	−1	(ab odd)

There is an easy way to fix this vulnerability. Instead of using the group $\langle g \rangle$, use the group $\langle g^2 \rangle$. All the elements in $\langle g^2 \rangle$ have a square root.

Homomorphic weaknesses

Homomorphic weakness attack (on RSA)

An insecure scheme...

RSA could be used as a signature scheme, where “signing” is encryption with the secret key, and “verifying” is decryption with the public key.

$\text{Sign}_{s_k}(m)$: Given $m, \langle N, d \rangle$, output: $(m, s) \leftarrow (m, m^d \bmod N)$

$\text{Vrfy}_{p_k}(m, s)$: Given $(m, s), \langle N, e \rangle$, if $m = s^e \bmod N$ then valid.

The above is not secure at all. There is a common misconception that a secure signature scheme can be achieved via “encryption” in this way:

1. Signing is done by “encrypting” the message using private key,
2. Verification is done by “decrypting” the signature using public key. [a]

This technique might not be secure. (Some textbooks use the term “encryption/decryption” in describing signature schemes. This is very misleading). Perhaps you can consider the effect of homomorphic properties on this scheme.

[a] RSA has an interesting property that we can use private key for encryption and public key for decryption. This is not true for other PKC.

Attacks #4

Exponentiation timing

Timing attacks (on exponentiation)

Efficient exponentiation

The first example of a timing attack on exponentiation (as in RSA) was introduced by Paul Kocher in 1995, while he was an undergraduate at Stanford. The attack attempts to find the key k, and exploits how efficient exponentiation is carried out.

Note that RSA does repeated multiplication, and Elgamal uses point addition, an analog of exponentiation. The algorithms use doubling and look the same:

RSA

```
To compute c^d mod N in RSA
Q = 1;
foreach bit in key d {
    Q = Q*Q mod N;
    if (bit==1) {
        Q = Q*c mod N;
    }
}
return Q;
```

ECC

```
To compute kP in ECC
Q = O;
foreach bit in key k {
    Q = 2Q;
    if (bit==1) {
        Q = Q + P;
    }
}
return Q;
```

Fault injection

Fault injection attack (in RSA, if using CRT)

What can the attacker do?

Assume the attacker can obtain the output $m = (ae_1 + be_2) \bmod N$.

The attacker can introduce noise/errors during the computation of e_1, and no error during the computation of e_2. Hence the attacker obtains

$$m' = (ae_1' + be_2) \bmod N$$

Now, the attacker computes $\gcd(m–m', N)$. The gcd is either p or q.

Why does it work?

Note that $m - m' = a(e_1 - e_1') \bmod N$, and that $a = 0 \bmod q$ (but is not zero). If $m–m'$ is not divisible by p, then

$$\gcd(m - m', N) = q$$

The exam, and the exam paper

The exam: Please attend :)

- Tuesday, 22nd November 2022, in LT15, and SR2 (COM1 #02-04)
- In the afternoon - 1:00 - come early, as you will have to sign in...
- You can bring in textbooks/papers, but be cautious - the answers will not be in them.

The exam paper: Quick pointers...

- Write in the boxes provided in the question paper, neatly.
- If you have to go outside the boxes, write on the backs of the page.
- Pencil, pen both OK - as long as it is clear.
- Watch your time on each question.
- 10-15% of the exam could be hard.
- Last year's exam, is a good guide to the content of the exam.

Weighting for questions

Multiple questions in each section...

Question sections	This Year	Last Year
General topics, short answers	Q1 (8)	Q1 (8)
HASH and MAC	Q2 (8)	Q2 (10)
Symmetric encryption	Q3 (8)	Q3 (7)
Asymmetric encryption, signatures	Q4 (8)	Q4 (8)
Secrets and MPC	Q5 (8)	Q5 (7)
Total:	Q1-5 (40)	Q1-5 (40)

There is no need for a calculator. There are no questions that require a calculator - any arithmetic that is there is pretty easily done.

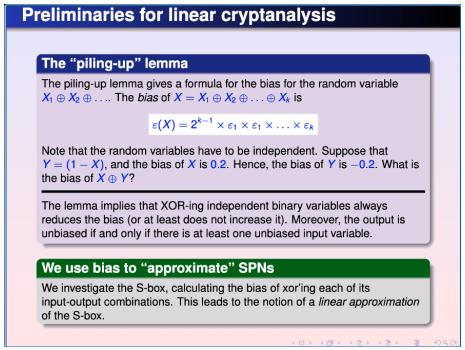
What to expect

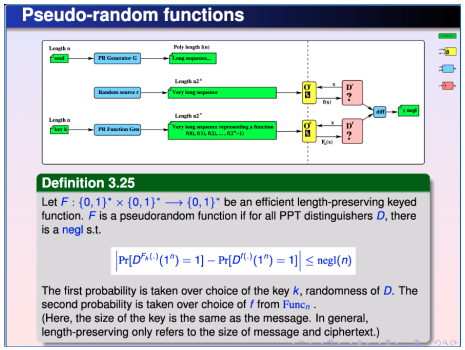
1: General topics, short answers (8)

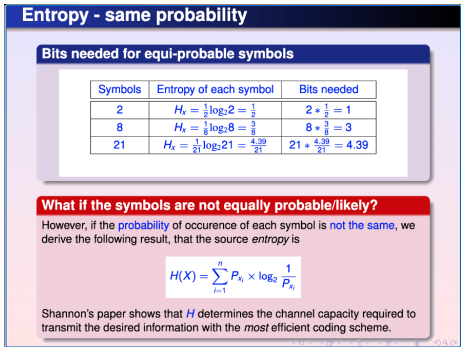
These could cover any topic, including both topics mentioned in class, and topics you should have run into.

---

5 questions, 8 marks. 2 or 3 lines for each question.







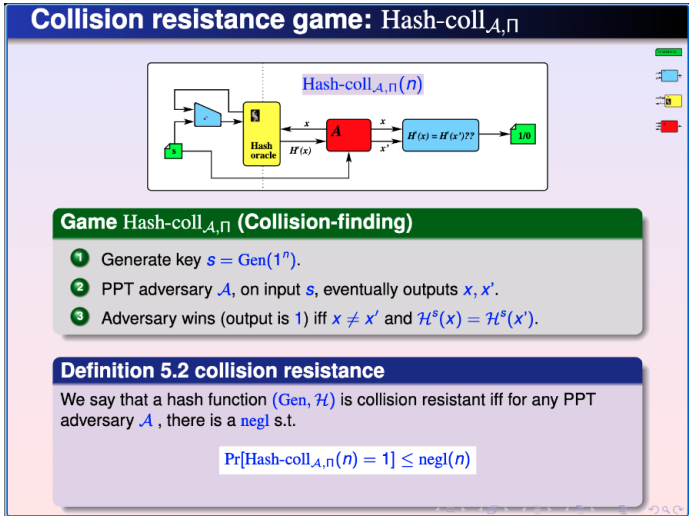
What to expect

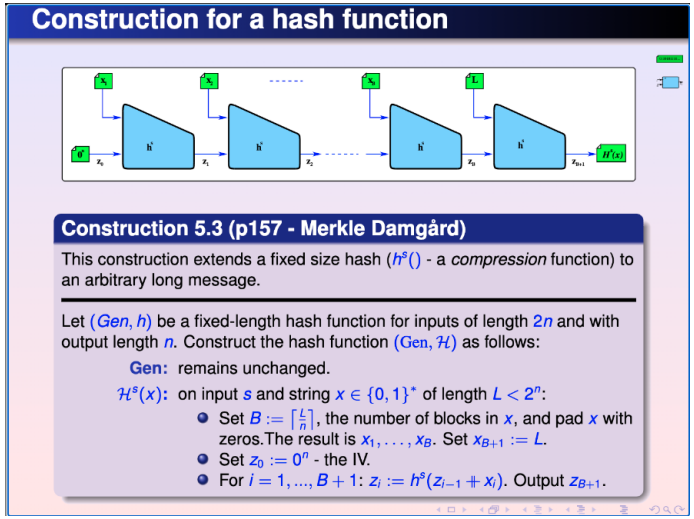
2: Hash and MAC (8)

In class we looked at MAC and hashes, and how they were interpreted. Think properties, proof of properties, proof by example, attacks, ways in which they are used in other constructions.

---

3 questions, 8 marks. You might have to write more or show a sketch proof or something for these questions.





What to expect

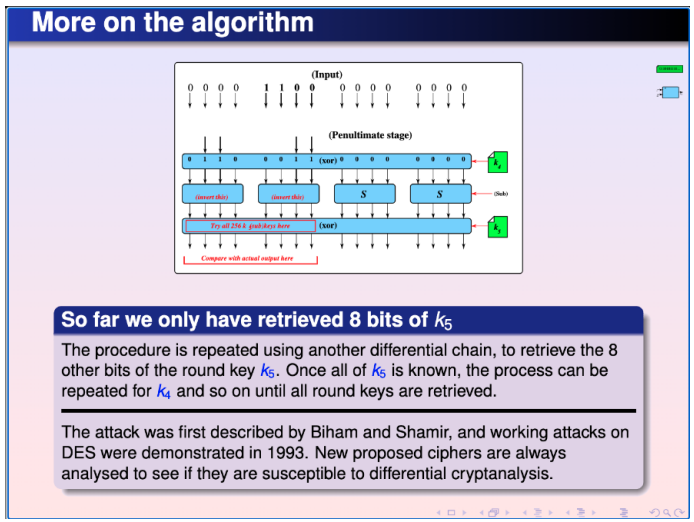
3: Symmetric encryption (8)

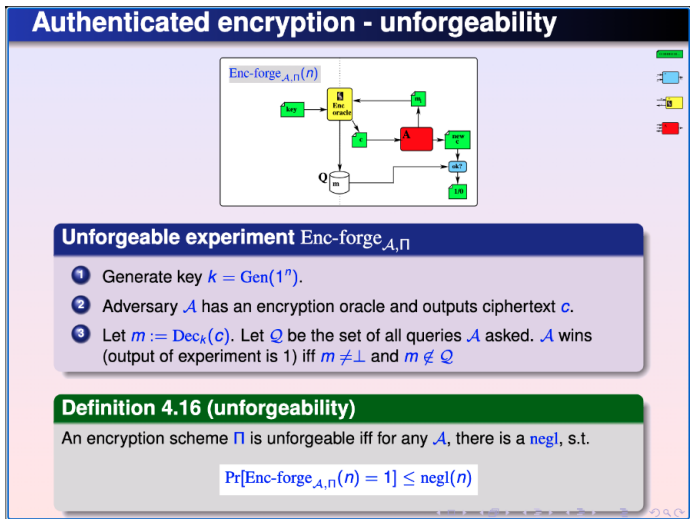
You are expected to be able to

- Comment on cryptographic issues related to symmetric schemes.
- Give technical comments on the types of symmetric schemes.
- Give comments on attacks on symmetric encryption schemes.

---

3 questions, 8 marks. You might have to write more for these questions.





What to expect

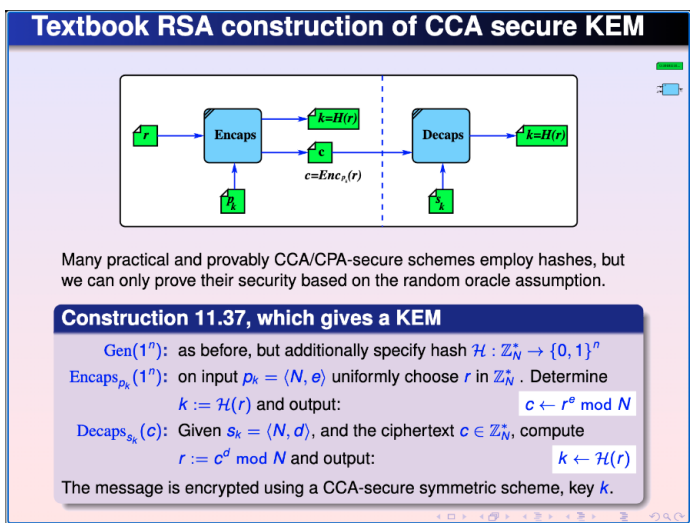
4: Asymmetric encryption, signatures (8)

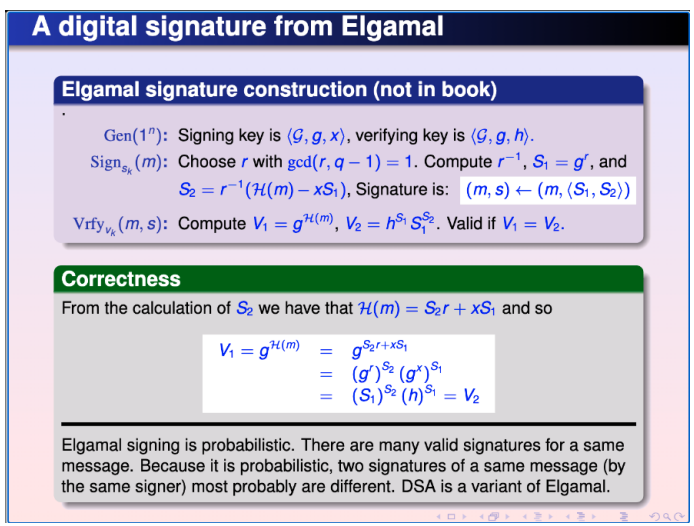
You are expected to be able to

- Comment on cryptographic issues related to asymmetric encryption and signature schemes.
- Give technical comments on the types of asymmetric schemes.
- Give comments on attacks on asymmetric encryption schemes.
- Understand and evaluate asymmetric constructions.

---

3 questions, 8 marks. You might have to write more for these questions.





What to expect

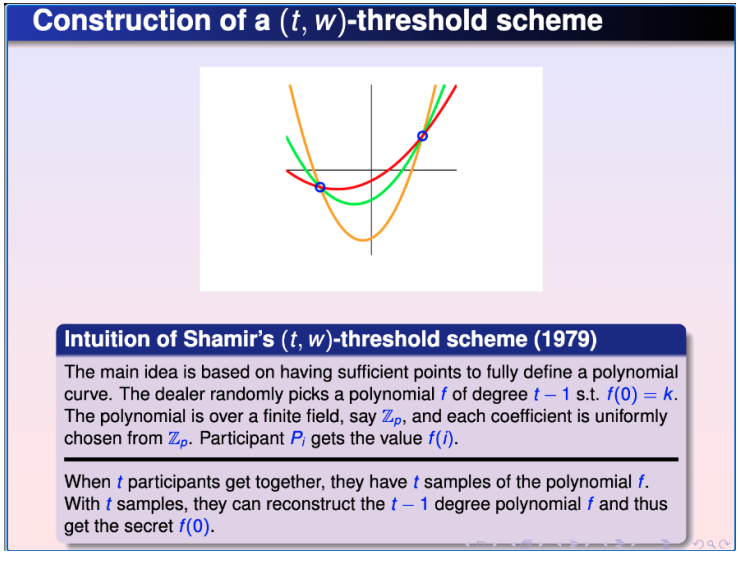
5: Secrets and MPC (8)

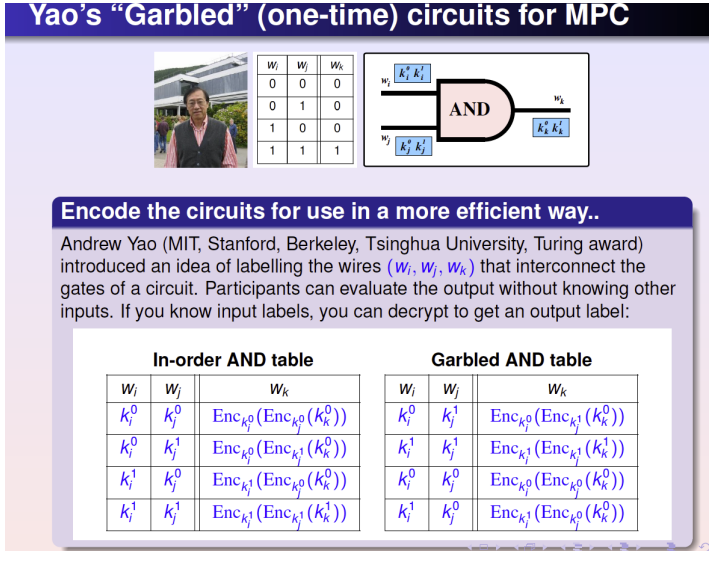
You are expected to be able to

- Understand notions of secret sharing.
- Understand ideas in MPC.

---

3 questions, 8 marks. You might have to write more for these questions.





Once again, with feeling...

I really hope I have shown you a few things that are interesting to you, and that help you in the future... I will be around up until the exam for help.

Thank you!