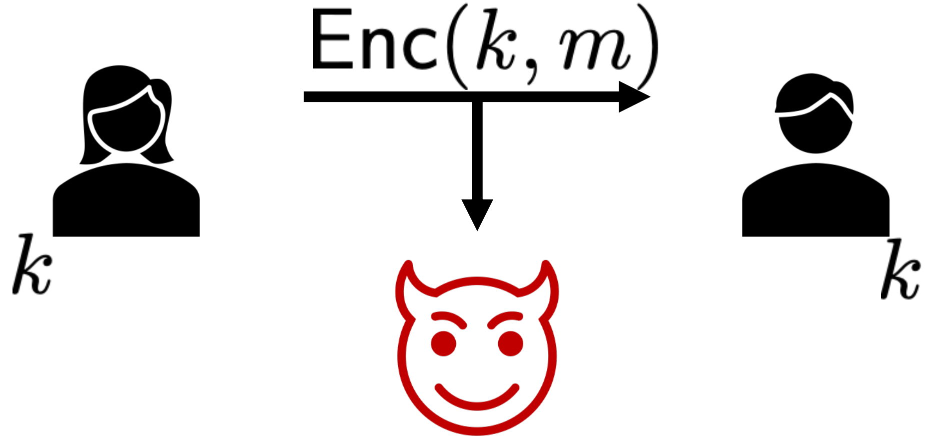
CS 258: Quantum Cryptography

Mark Zhandry

Pre-modern Cryptography (2000 BC – mid 1900's AD)

Cryptography ≈ (symmetric) encryption



Serious usage limited mostly to state-level entities

Tug-of-war between code makers & breakers; breakers usually win

Modern Cryptography (Mid 1900's – Present)

Cryptography =

(Public key) Encryption

Attribute-based encryption

Digital signatures

Proofs of knowledge

Zero knowledge

Homomorphic encryption

Digital money

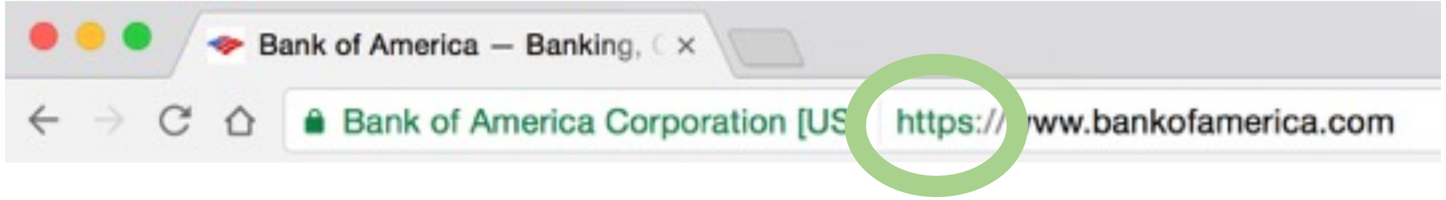
Program Obfuscation

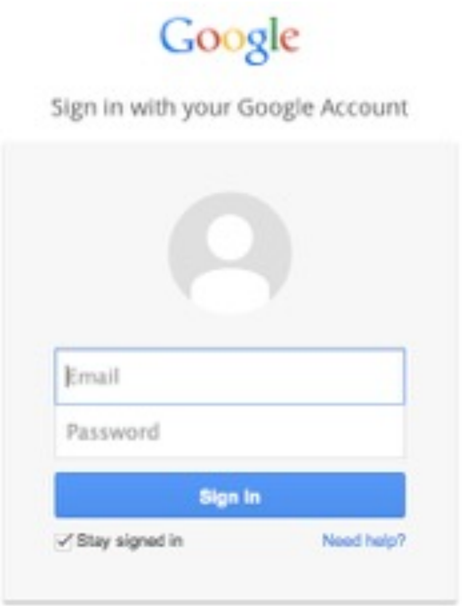
Traitor tracing

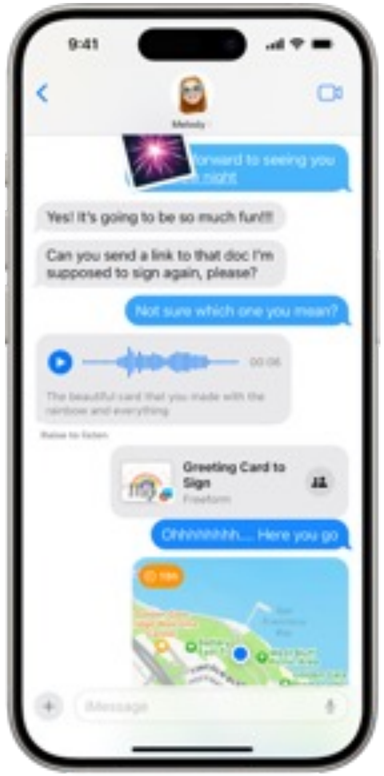
• • •

Modern Cryptography (Mid 1900's – Present)

Cryptography is everywhere







Modern Cryptography (Mid 1900's – Present)

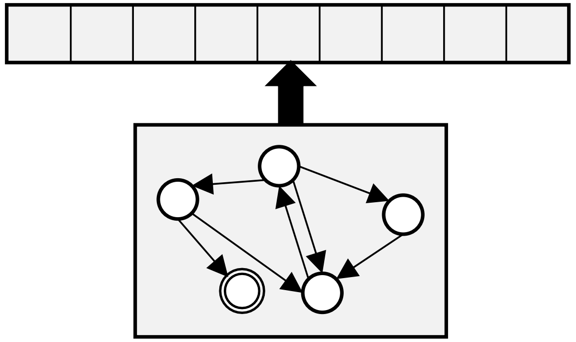
Cryptography almost never fails in the real world, because we "prove" it is secure

However, we are on the precipice of another major shift in cryptography due to *quantum computers*

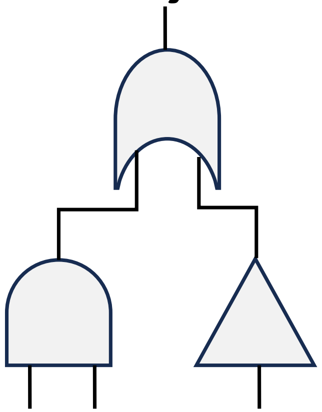
As we will discuss later in this lecture, cryptography relies on computational problems that are intractable for efficient computation

What is "efficient" computation?

1900's – Present: can run *efficiently* on *today's* computers



Turing machines

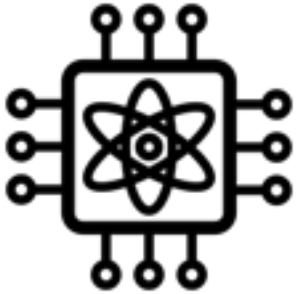


(Classical) circuits

(Extended) Church-Turing Thesis: Today's computers can (efficiently) compute anything that can be (efficiently) computed by *any* physical process

What is "efficient" computation?

The future: can run *efficiently* on *quantum* computers



(Extended) Church-Turing Thesis: Today's computers can (efficiently) compute anything that can be (efficiently) computed by *any* physical process

What does quantum computing mean for cryptography?

Quantum Cryptanalysis: All currently-deployed public key cryptography will be broken

Post-quantum cryptography: developing new (classical) protocols that are secure against quantum computers

- Must start now to protect against quantum “harvest-now-decrypt-later” attacks
- Requires revisiting the entire theory of modern classical cryptography

Quantum cryptography: developing new *quantum* protocols that achieve never-before-possible capabilities

This Course

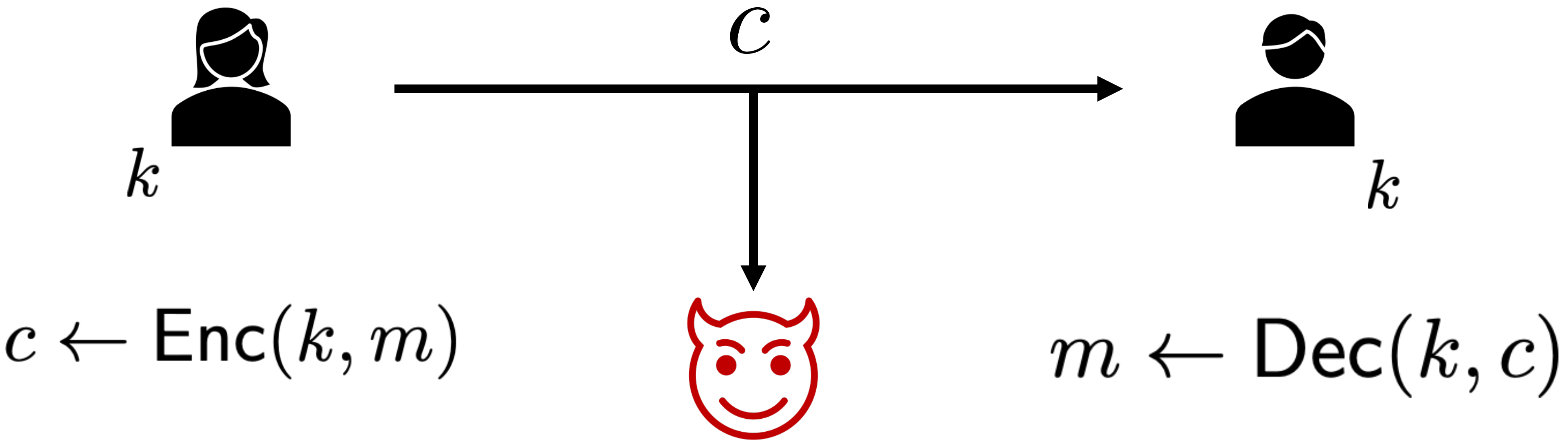
Overview of **quantum cryptanalysis**, and **post-quantum** and **quantum** cryptography

Prerequisites: Knowledge of linear algebra and algorithms
(No prior knowledge of cryptography or quantum is assumed)

Brief background of
classical cryptography

For now, focus on *encryption*

Symmetric Encryption

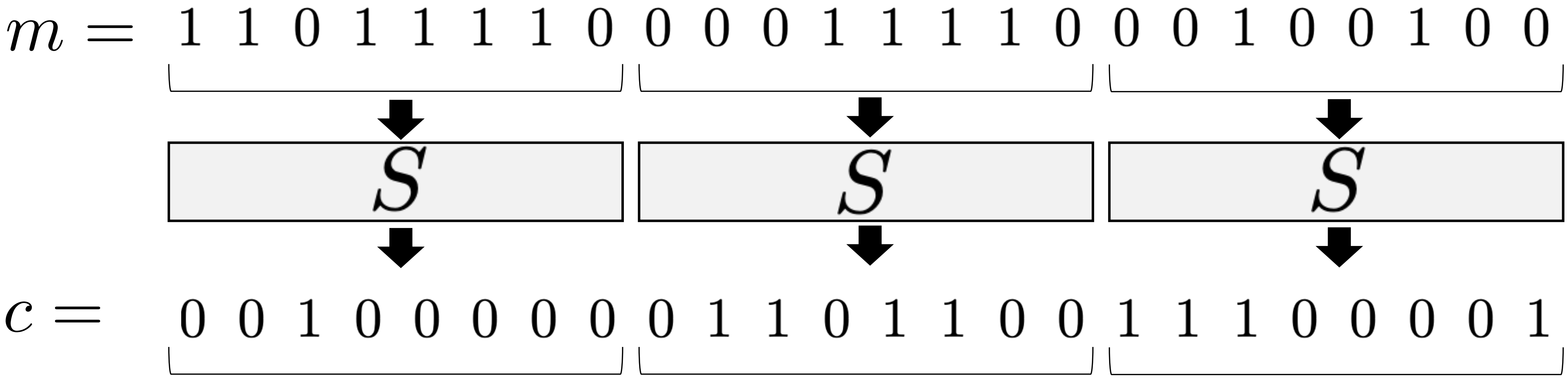


"learns nothing" about m

Kerckhoff's Principle: assume $\mathsf{Enc}, \mathsf{Dec}$ are public knowledge, only k kept secret

How do we build symmetric encryption?

Substitutions:



k determines S

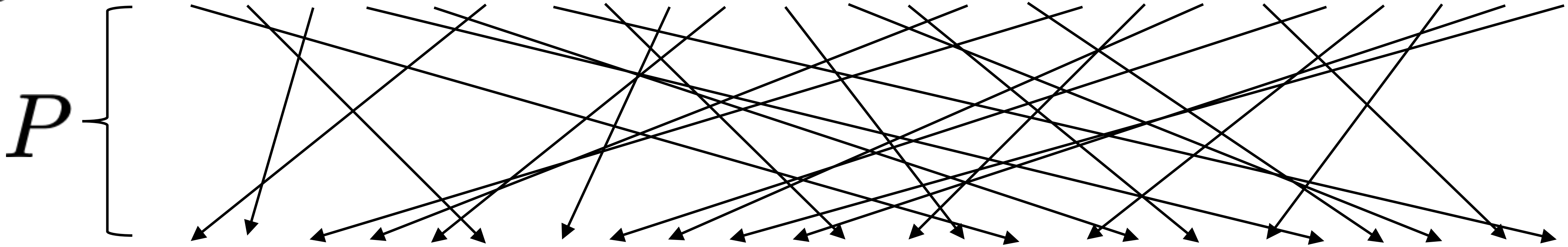
How do we build symmetric encryption?

Substitutions broken by frequency analysis:
Most common byte is probably an “e”, second most is probably a “t”, etc.

How do we build symmetric encryption?

Permutations:

$$m = 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0$$



$$c = 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$$

k determines P

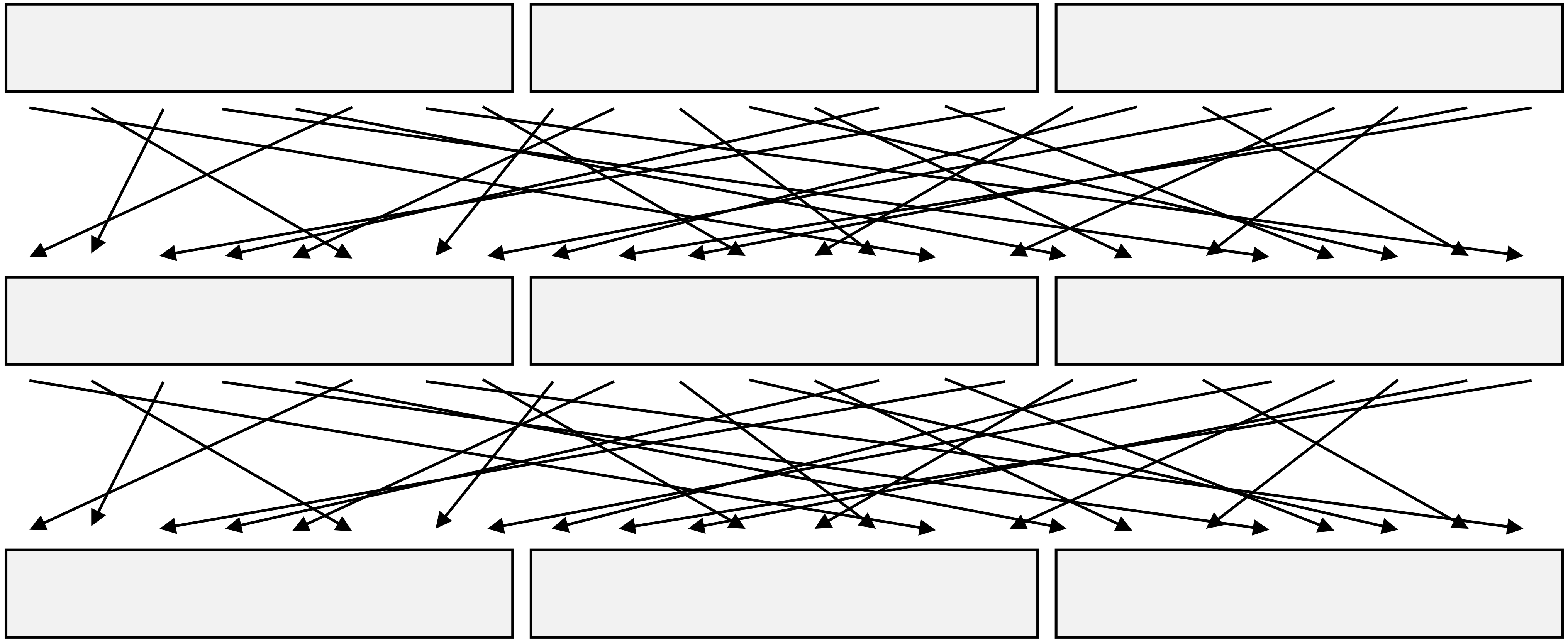
How do we build symmetric encryption?

Permutations broken by numerous methods:

- Number of 1's revealed
- In order to keep description of P small, it has extra structure, which can lead to breaks

How do we build symmetric encryption?

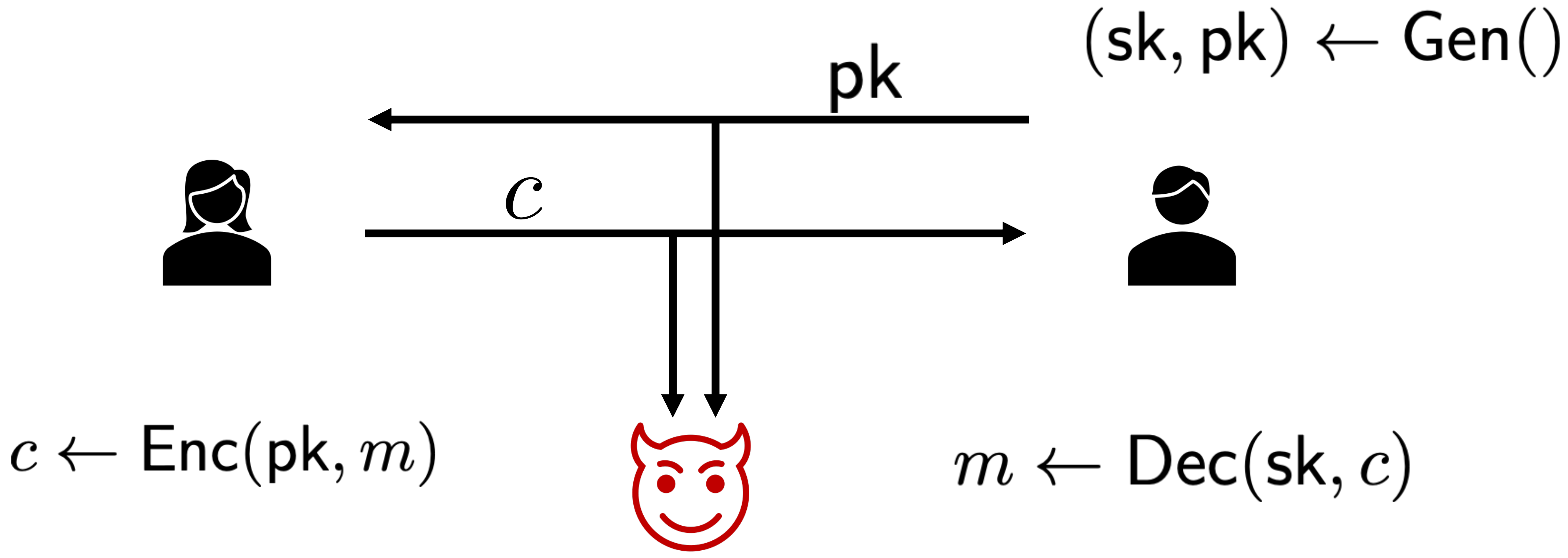
Substitution-Permutation Network



It works! Basis for many modern symmetric encryption schemes

Fundamental limitation of symmetric encryption:
how to Alice and Bob share k in the first place?

Asymmetric (or Public Key) Encryption



"learns nothing" about m

How do we build public key encryption?

ElGamal (Toy Version):

λ typically ≈ 2000

$\mathsf{Gen}(1^\lambda)$: Choose a random λ-bit prime p

Choose random generator g of $\mathbb{Z}_p^*$

Choose random $\alpha \leftarrow \{0, 1, 2, \cdots, p-2\}$

Let $h = g^\alpha \bmod p$

$\mathsf{sk} = (p, g, \alpha)$ $\mathsf{pk} = (p, g, h)$

How do we build public key encryption?

ElGamal (Toy Version):

$\text{Enc}(\ (p, g, h)\ , m)$:

Interpret m as an element of $\mathbb{Z}_p^*$

Choose random $\beta \leftarrow \{0, 1, 2, \cdots, p-2\}$

Let $u = g^\beta \bmod p \quad v = h^\beta \times m \bmod p$

Output $c = (u, v)$

How do we build public key encryption?

ElGamal (Toy Version):

$\mathsf{Dec}(\ (p, g, \alpha)\ ,\ (u, v)\ )$:

Output $m = v/u^{\alpha} \bmod p$

Correctness:

$$v/u^{\alpha} = (h^{\beta}m)/(g^{\beta})^{\alpha} = (g^{\alpha\beta}m)/g^{\alpha\beta} = m$$

What does it mean that an eavesdropper should “learn nothing” about the message?

Attempt 1: Statistical Security

Intuitive definition: view of adversary "contains no information" about m

Attempt 1: Statistical Security

Problem: useful schemes cannot be statistically secure

Consider public key in ElGamal $p, g, h = g^\alpha \bmod p$

Simple algorithm to compute α:

For $\alpha' = 0, 1, 2, \cdots, p-2$:
    If $g^{\alpha'} \bmod p = h$, output α'

Brute-Force Search

Try all possibilities until you find the right one

Note: need to be able to tell if you got the right one

Brute-Force Search

Brute-force search always possible for PKE

Brute-force search always possible for SKE,
assuming total length of messages sent >> length of key

One-Time Pad

$$\mathsf{Enc}(k, m) = k \oplus m$$

$$\mathsf{Dec}(k, c) = k \oplus c$$

$$k \oplus (k \oplus m) = m$$

No way to check if guessed key is correct, if encrypting single message

Almost all cryptography can be broken via brute-force search

What do we do?

Solution: Computational Security

Notice that a brute-force search
takes a huge about of time

ElGamal with 2000-bit prime: 2^{2000} trials

Every particle in visible universe replaced
with the world's fastest supercomputer ➡ 2^{1615} years

Compare to life of universe: 2^{34} years

Solution: Computational Security

Notice that a brute-force search
takes a huge about of time

Only ask for security against
“efficient” adversaries

What is efficient?

In practice:

Time $\leq 2^{128}, 2^{256}$

Total bitcoin network:

$\approx 2^{100}$ operations/year

In theory:

Polynomial time

Beating Brute-Force Search

We can always make brute-force intractable by making keys long

However, brute-force may not be fastest algorithm

E.g. best attacks on ElGamal run in time $2^{O((\log p)^{1/3}(\log \log p)^{2/3})} \ll p$

Beating Brute-Force Search

Rule-of-thumb:

Symmetric crypto: due to lack of mathematical structure, best attacks typically run in time 2^n

Public key crypto: Depends on underlying math, hope to get as close to 2^n as possible

Cryptography and P vs NP

Polynomial-time adversaries ➡ Adversary $\in$ ~~P~~ BPP

(allow adversary random coins)

Brute-force possible ➡ Breaking scheme is in NP

Therefore, (most) cryptography can only exist if $P \neq NP$

(or even $NP \not\subseteq BPP$)

Cryptography and P vs NP

As a consequence, (almost) all cryptosystems rely on unproven computational assumptions

Neet at least $P \neq NP$, usually much more

The Fundamental Formula of Modern Cryptography

Secure Cryptosystem =

Protocol

+

Formal Security Model **M** — Usually conservative modeling of adversary's capabilities

+

Computational Assumption **P** — Widely studied, concrete assumptions

+

Proof that **P** implies **M** — Breaking **M** at least as hard as solving **P**

Example: proving the security of ElGamal

Step 1: Define Public Key Encryption

Step 1a: Define Syntax, Correctness

Def (PKE, syntax): A public key encryption scheme is a triple of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ satisfying the following:

- $\mathsf{Gen}(1^\lambda)$: probabilistic polynomial-time (classical) procedure which takes as input a security parameter λ (represented in unary), and samples a secret/key public pair $(\mathsf{sk}, \mathsf{pk})$
- $\mathsf{Enc}(\mathsf{pk}, m)$: PPT procedure which takes as input the public key pk and message m, and samples a ciphertext c
- $\mathsf{Dec}(\mathsf{sk}, c)$: Deterministic PT procedure which takes as input the secret key sk and ciphertext c, and outputs a message m
- **Correctness:** $\forall \lambda, (\mathsf{sk}, \mathsf{pk})$ in support of $\mathsf{Gen}(1^\lambda), \forall m \in \{0,1\}^*$

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1$$

The Security Parameter

Allow for tuning security level of protocol

In practice, often only a couple parameters standardized (e.g. 128,256)

In theory, can be any natural number; necessary for defining "polynomial time"

Represented in unary so that $\mathsf{Gen}(1^\lambda)$ runs in time $\mathsf{poly}(\lambda)$

Probabilistic algorithms

Gen is probabilistic so that each run gives different keys

remember that the algorithm Gen is publicly known (Kerckhoff's Principle)

Enc is probabilistic for security (see homework)

Dec is *deterministic* since it should always just output m

Correctness as a probability

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1$$

Pedantic note: need to wrap in probability since Enc is not a function

Def (PKE, syntax): A public key encryption scheme is a triple of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ satisfying the following:

- $\mathsf{Gen}(1^\lambda)$: probabilistic polynomial-time (classical) procedure which takes as input a security parameter λ (represented in unary), and samples a secret/key public pair $(\mathsf{sk}, \mathsf{pk})$
- $\mathsf{Enc}(\mathsf{pk}, m)$: PPT procedure which takes as input the public key pk and message m, and samples a ciphertext c
- $\mathsf{Dec}(\mathsf{sk}, c)$: Deterministic PT procedure which takes as input the secret key sk and ciphertext c, and outputs a message m
- **Correctness:** $\forall \lambda$, $(\mathsf{sk}, \mathsf{pk})$ in support of $\mathsf{Gen}(1^\lambda)$, $\forall m \in \{0,1\}^*$

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1$$

Step 1b: Define Security

“Negligible”

In practice: $\leq 2^{-128}, 2^{-256}$

In theory:

Def (negligible): A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is *negligible* if, for all polynomials p, $\exists N_p \in \mathbb{N}$ such that for all $\lambda \geq N_p$,

$$f(\lambda) \leq 1/p(\lambda)$$

A function that is not negligible is called *non-negligible*

Def (PKE, security): A PKE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is indistinguishable under a chosen plaintext attack (IND-CPA-secure, or just CPA-secure) if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that

$$|\Pr[W_0(\lambda)] - \Pr[W_1(\lambda)]| \leq \epsilon(\lambda)$$

where $W_b(\lambda)$ is the event that $\mathcal{A}$ outputs 1 in the following:

- Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$, give pk to $\mathcal{A}$
- $\mathcal{A}$ produces two msgs $m_0, m_1 \in \{0,1\}^*$ *of the same length*
- Run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ and give c to $\mathcal{A}$
- $\mathcal{A}$ outputs an output guess $b' \in \{0,1\}$

CPA security is conservative

CPA-security says that the adversary knows everything about the message except a single bit, and must learn that bit

The adversary may even choose everything about the message, except for the bit it is trying to learn

In real life, adversary may influence message, and may have side information, but unlikely to be *that* strong

By having a conservative definition, we don't need to worry about exact abilities, and know we have security regardless

Restriction that m_0, m_1 have the same length is (unfortunately) necessary, since ciphertext length is revealed

Otherwise, “Hello” vs, say, an entire movie would have ciphertexts of the same length

Def (PKE, security): A PKE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is indistinguishable under a chosen plaintext attack (IND-CPA-secure, or just CPA-secure) if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that

$$|\Pr[W_0(\lambda)] - \Pr[W_1(\lambda)]| \leq \epsilon(\lambda)$$

where $W_b(\lambda)$ is the event that $\mathcal{A}$ outputs 1 in the following:

- Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$, give pk to $\mathcal{A}$
- $\mathcal{A}$ produces two msgs $m_0, m_1 \in \{0,1\}^*$ *of the same length*
- Run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ and give c to $\mathcal{A}$
- $\mathcal{A}$ outputs an output guess $b' \in \{0,1\}$

Step 2: Specify Protocol

How do we build public key encryption?

$\mathsf{Gen}(1^\lambda)$: Choose a random λ-bit prime p

Choose random generator g of $\mathbb{Z}_p^*$

Choose random $\alpha \leftarrow \{0, 1, 2, \cdots, p-2\}$

Let $h = g^\alpha \bmod p$

$\mathsf{sk} = (p, g, \alpha)$ $\mathsf{pk} = (p, g, h)$

How do we build public key encryption?

Enc((p, g, h) , m) :

Interpret m as an element of $\mathbb{Z}_p^*$

Choose random $\beta \leftarrow \{0, 1, 2, \cdots, p-2\}$

Let $u = g^\beta \bmod p$ $\quad v = h^\beta \times m \bmod p$

Output $c = (u, v)$

How do we build public key encryption?

$\text{Dec}(\ (p, g, \alpha)\ ,\ (u, v)\ ):$

Output $m = v/u^\alpha \bmod p$

Lemma: Toy ElGamal is a PKE scheme

Proof: All algorithms polynomial time. Correctness:

$$v/u^\alpha = (h^\beta m)/(g^\beta)^\alpha = (g^{\alpha\beta} m)/g^{\alpha\beta} = m$$

Step 3: State assumptions

Assumption (Discrete Log): For any PPT algorithm $\mathcal{A}$, there exists a negligible function ϵ such that

$$\Pr[\mathcal{A}(p, g, h) = \alpha] \leq \epsilon(\lambda)$$

where:

- p is a random λ-bit prime
- g is a random generator of $\mathbb{Z}_p^*$
- $\alpha \leftarrow \{0, 1, 2, \cdots, p-2\}$ is random

Necessary, but not necessarily sufficient for ElGamal to be secure

Assumption (Decisional Diffie-Hellman): For any PPT algorithm $\mathcal{A}$, there exists a negligible function ϵ such that

$$\begin{aligned} &|\Pr[\mathcal{A}(p, g, g^\alpha \bmod p, g^\beta \bmod p, g^{\alpha\beta} \bmod p) = 1] \\ &\quad - \Pr[\mathcal{A}(p, g, g^\alpha \bmod p, g^\beta \bmod p, g^\gamma \bmod p) = 1]| \le \epsilon(\lambda) \end{aligned}$$

where:

- p is a random λ-bit prime
- g is a random generator of $\mathbb{Z}_p^*$
- $\alpha, \beta, \gamma \leftarrow \{0, 1, 2, \cdots, p-2\}$ are random

Despite decades of attempts at solving DDH, the best algorithms are sub-exponential time. The DDH assumption therefore is widely believed.

Step 4: Prove Security

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof: Let $\mathcal{A}$ be a supposed adversary for the CPA-security of ElGamal.

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof:

Define $W_b(\lambda)$ as the event that $\mathcal{A}$ outputs 1 in the following:

- Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$, give pk to $\mathcal{A}$
- $\mathcal{A}$ produces two msgs m_0, m_1
- Run $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ and give c to $\mathcal{A}$
- $\mathcal{A}$ outputs an output guess $b' \in \{0, 1\}$

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof:

Define $W_b(\lambda)$ as the event that $\mathcal{A}$ outputs 1 in the following:

- Run $\mathsf{pk} = (p, g, h)$ and give pk to $\mathcal{A}$, where...
- $\mathcal{A}$ produces two msgs m_0, m_1
- Give $c = (u, v)$ to $\mathcal{A}$ where $u = g^\beta \bmod p$, $v = h^\beta \times m_b \bmod p$
- $\mathcal{A}$ outputs an output guess $b' \in \{0, 1\}$

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof:

Define $W_b(\lambda)$ as the event that $\mathcal{A}$ outputs 1 in the following:

- Run $\mathsf{pk} = (p, g, h)$ and give pk to $\mathcal{A}$, where…
- $\mathcal{A}$ produces two msgs m_0, m_1
- Give $c = (u, v)$ to $\mathcal{A}$ where $u = g^{\beta} \bmod p$, $v = g^{\alpha\beta} \times m_b \bmod p$
- $\mathcal{A}$ outputs an output guess $b' \in \{0, 1\}$

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof: Our goal is to prove that

$$|\Pr[W_0(\lambda)] - \Pr[W_1(\lambda)]| \leq \epsilon(\lambda)$$

for some negligible function ϵ

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof:

Define $\boxed{V_b(\lambda)}$ as the event that $\mathcal{A}$ outputs 1 in the following:

- Run $\mathsf{pk} = (p, g, h)$ and give pk to $\mathcal{A}$, where…
- $\mathcal{A}$ produces two msgs m_0, m_1
- Give $c = (u, v)$ to $\mathcal{A}$ where $u = g^\beta \bmod p$, $\boxed{v = g^\gamma \times m_b \bmod p}$
- $\mathcal{A}$ outputs an output guess $b' \in \{0, 1\}$

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof:

$$\begin{aligned} |\Pr[W_0(\lambda)] - \Pr[W_1(\lambda)]| \leq & \, |\Pr[W_0(\lambda)] - \Pr[V_0(\lambda)]| \\ & + |\Pr[V_0(\lambda)] - \Pr[V_1(\lambda)]| \\ & + |\Pr[V_1(\lambda)] - \Pr[W_1(\lambda)]| \end{aligned}$$

Now we will bound each term separately

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof: $|\Pr[W_0(\lambda)] - \Pr[V_0(\lambda)]|$:

Let $\mathcal{B}(p, g, A, B, C)$ be the following DDH adversary:

- Give $\mathsf{pk} = (p, g, h = A)$ to $\mathcal{A}$
- When $\mathcal{A}$ produces two messages m_0, m_1, reply with $c = (u = B, v = C \times m_0 \bmod p)$
- Output whatever $\mathcal{A}$ outputs

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof: $|\Pr[W_0(\lambda)] - \Pr[V_0(\lambda)]|$:

Observe that $|\Pr[W_0(\lambda)] - \Pr[V_0(\lambda)]| =$

$$|\Pr[\mathcal{B}(p, g, g^\alpha \bmod p, g^\beta \bmod p, g^{\alpha\beta} \bmod p) = 1]$$
$$- |\Pr[\mathcal{B}(p, g, g^\alpha \bmod p, g^\beta \bmod p, g^\gamma \bmod p) = 1]$$

which by DDH must be at most a negligible $\epsilon_0(\lambda)$

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof: $|\Pr[V_0(\lambda)] - \Pr[V_1(\lambda)]|$:

Only difference:

$v = g^\gamma \times m_0 \bmod p$ vs $v = g^\gamma \times m_1 \bmod p$

g^γ is uniform in $\mathbb{Z}_p^*$ ➡ $g^\gamma \times m_0 \bmod p$, $g^\gamma \times m_1 \bmod p$ are uniform

➡ $\Pr[V_0(\lambda)] = \Pr[V_1(\lambda)]$

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof: $|\Pr[V_1(\lambda)] - \Pr[W_1(\lambda)]|$:

By analogous arguments,

$$|\Pr[V_1(\lambda)] - \Pr[W_1(\lambda)]| \leq \epsilon_1(\lambda)$$

for some negligible ϵ_1

Theorem: Assuming DDH, ElGamal is CPA-secure

Proof:

$$\begin{aligned} |\Pr[W_0(\lambda)] - \Pr[W_1(\lambda)]| &\leq |\Pr[W_0(\lambda)] - \Pr[V_0(\lambda)]| \\ &\quad + |\Pr[V_0(\lambda)] - \Pr[V_1(\lambda)]| \\ &\quad + |\Pr[V_1(\lambda)] - \Pr[W_1(\lambda)]| \\ &\leq \epsilon_0(\lambda) + \epsilon_1(\lambda) \end{aligned}$$

Sum of negligible funcs is negligible ■

Up Next: Quantum

The Fundamental Formula of Modern Cryptography

Secure Cryptosystem

=

Protocol

+

Formal Security Model **M**

+

Computational Assumption **P**

+

Proof that **P** implies **M**

All of these fundamentally assume classical adversaries

Need to revisit everything with quantum computers