Modern, Functional Cryptography

Jose Cardona

AiMe GmbH, York University

LambdaConf 2018

About Me

- I'm a software engineer at a small startup called AiMe GmbH
- OSS Contributor to a chunk of the FP ecosystem in Scala: Http4s, fs2, Scalaz
- Author of a Cryptography/Security library (work in progress!) called TSec
- Final (4th) year Honors Computer Security undergrad at York university in Toronto, Ontario

Big Thank you to the following people leading up to this talk:

- Professor Patrick Ingram, York University.
- Mohammad Reza Faghani.
- Navid Mohaghegh.
- John DeGoes and the LC crew for making it happen
- My OSS buddies: Edmund Noble, Alexander Konovalov, Emily Pillmore and Harrison Houghton for being top memers.

About This Workshop

Disclaimer: This is a foundations workshop. We are going to be covering topics as if you've never seen any cryptography, even though some or most of you have interacted with it in some way. I will aim, with this talk, even if you've had some contact with cryptography before, to better understand it with reasonable depth.

Why learn it with depth?

Why should we give a crap about learning about cryptography in depth?
You might think: "I'll never implement this" or ever have to deal with it.
"That's for the security teams to worry about!"

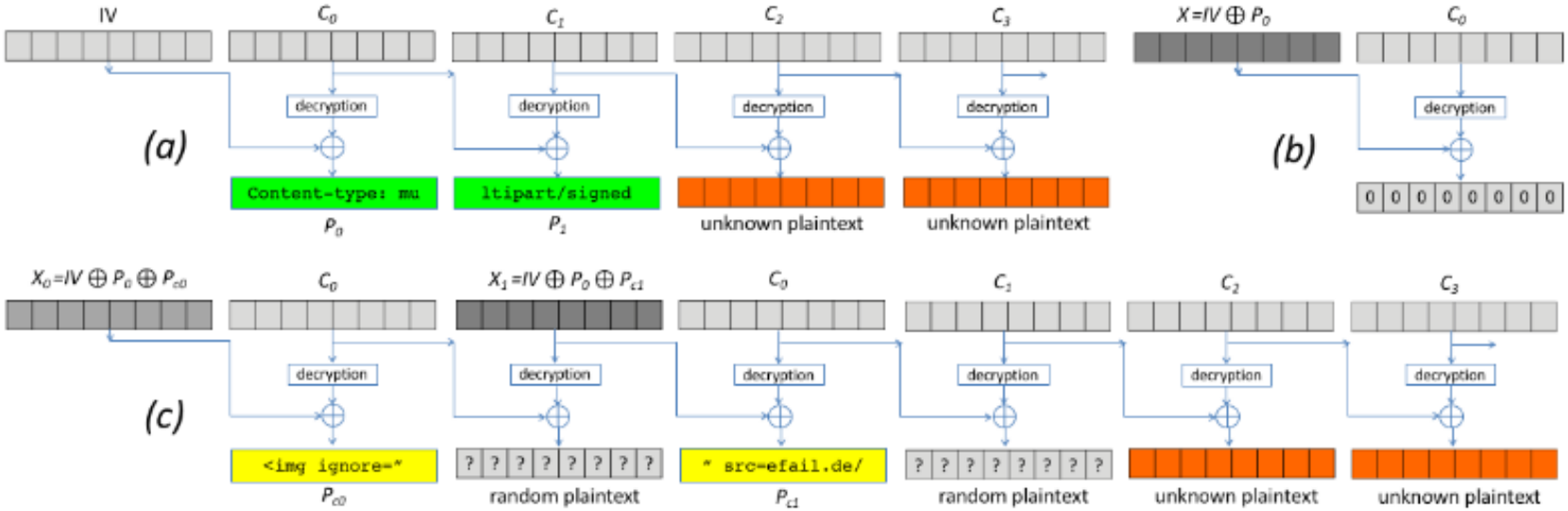
Real world examples: Padding oracles and POODLE

First published in 2002. Padding oracles are applied to CBC mode of operation on symmetric block ciphers. Padding is necessary on block ciphers due to fixed length operation.
Server leaks data about whether the padding of an encrypted message using a block cipher is correct.
Attacks using padding oracles have been as recent as 2016. A notable one in recent years has been POODLE:

- Publicly disclosed on October 2014
- Exploited downgrade functionality for interop in SSL 3.0

Real world examples: EFail

EFAIL describes vulnerabilities in the end-to-end encryption technologies OpenPGP and S/MIME that leak the plaintext of encrypted emails. !!ref needed here!!
The CBC/CFB gadget attack part of EFAIL vulnerabilities using a known plaintext attack to modify packets in an MITM fashion to extract the decrypted email information.



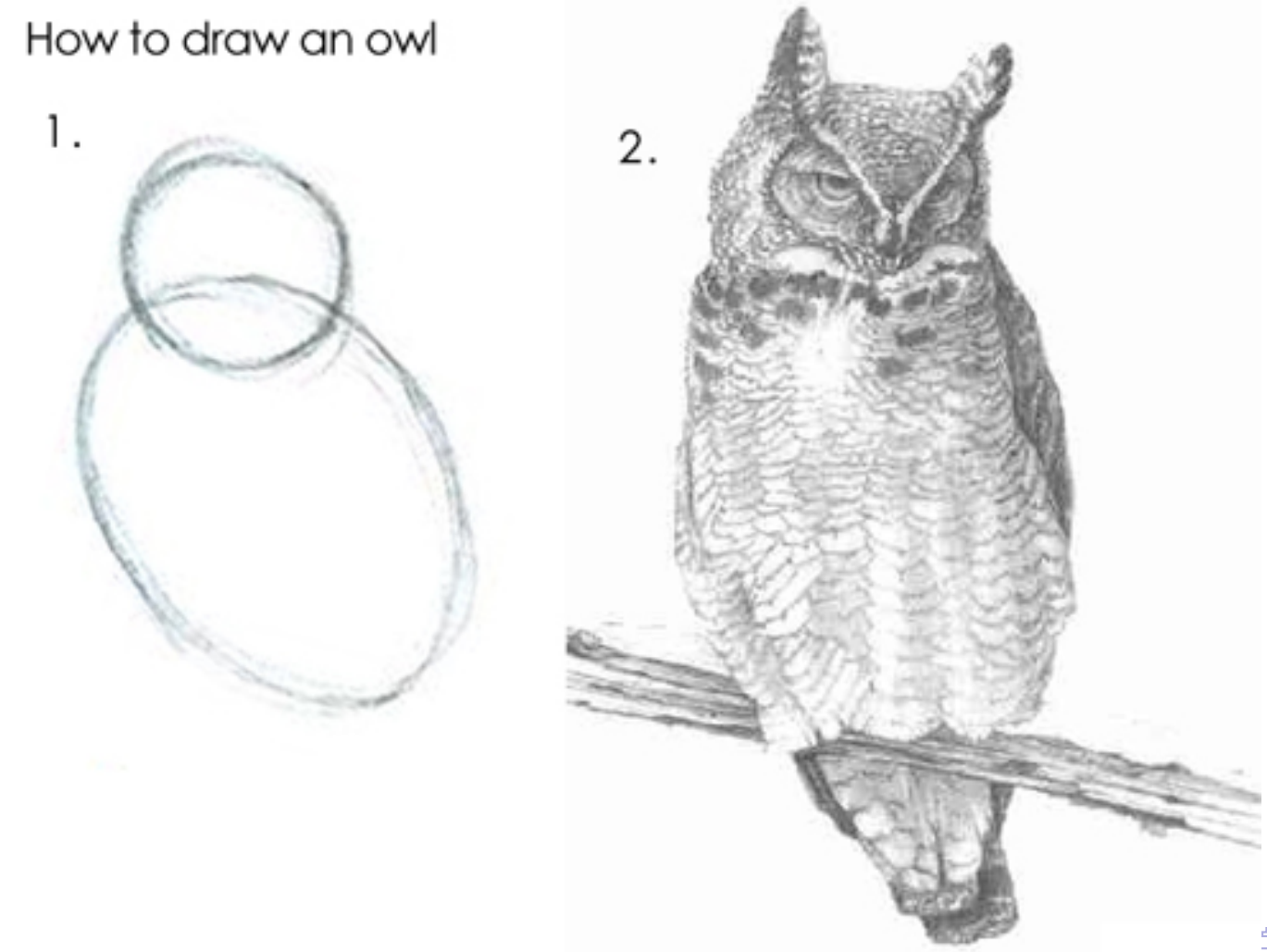
We will revisit this example at the end of symmetric ciphers.

What do these failures have in common?
Such failures were either implementation or fundamental understanding in what made something secure. It could've been at the time they were envisioned, like WEP which was one of the first ways we secured wifi, another can be more recent.
As developers, we have to be mindful of what we implement, and how we tackle cryptographic applications. As a rule of thumb, when it comes to primitives, despite anything I say in this talk, follow the golden rule:

- DON'T ROLL YOUR OWN CRYPTO

Before we dive in

This talk does not aim to give you neither a purely practical nor purely mathematical approach to cryptography. It's a little in between.
Our main enemy and limitation: Time! What I'm hoping to avoid:



The functional portion

I want to present to you cryptography from the perspective of how you should think of interacting with it in functional applications. That is: cryptographic primitives are things that are mathematically constructed, with (to some degree) proven "hardness", and thus follow certain mathematical properties.

To some degree, as we will see later, definition of things such as semantic security force us to not be able to state certain properties as "laws", so we don't have as strong of a notion of certain properties like we do other things such as programmatically stated natural transformations, monoidal or monadic composition (in the sense of something that has to happen), but we have strong, probabilistic notions of what should happen.

Resources

For those who after this talk would really like to learn more, here are a good chunk of sources:
Books:

- Introduction to Modern Cryptography: Katz and Lindell.
- Introduction to Mathematical Cryptography: Hoffstein, Piper and Silverman
- (Grad) Dan Boneh's Crypto book: http://toc.cryptobook.us/ (Free!)
- Security Engineering - Anderson.

Resources to play with:

- Cryptopals!!! https://cryptopals.com/
- Coursera Cryptography Course (Dan Boneh)
- Your company's assets and passwords (jk).

Introduction to Cryptography

Cryptography as a mathematical discipline can be thought of as a conglomerate containing:

- Number theory
- Abstract Algebra
- Mathematical Logic
- Combinatorics
- Information theory
- ... and more

Topics

Private-Key/Symmetric Cryptography

1. Ciphers, Computational security, modes of operation
2. Cryptographic Hash functions
3. Message Authentication

Public-Key/Asymmetric Cryptography:

1. Introduction to Number Theory, modular arithmetic and abstract algebra
2. Discrete Logarithms, Elgamal PKCS, Chinese Remainer Theorem, Attacks
3. RSA and Integer factorization
4. Digital Signatures
5. Elliptic Curve cryptography
6. Capstone Webapp

Introduction to Symmetric Cryptography

Back in the time of Julius Caesar, concern was being able to design codes to enable two parties to communicate in secret. Today, we call these codes "Encryption schemes".

The security of these classical schemes relied on a shared secret, or "key", known in advance between both parties but unknown to the eaves dropper. This scenario was the very basics of what grew to be private-key encryption.

Introduction to Symmetric Cryptography

In private key encryption, two parties with a shared key will aim to exchange information by one of the two scrambling or "encrypting" a message with the key, or "plaintext", to produce a ciphertext. The receiver, using the shared key, unscrambles or "decrypts" the ciphertext to recover the message.

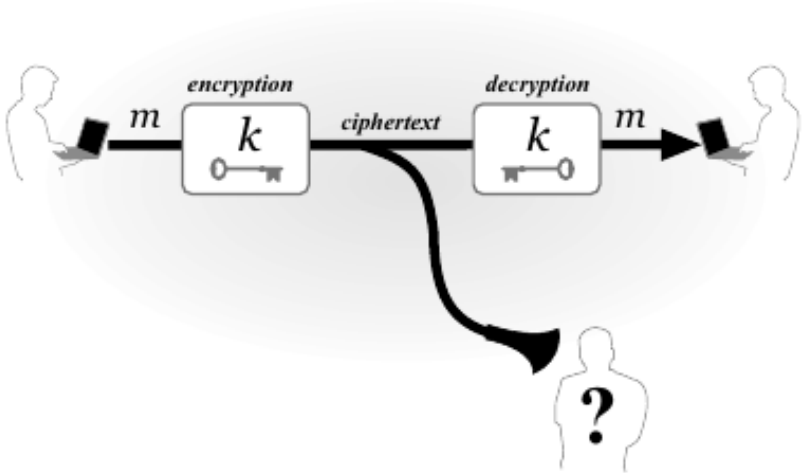


Figure: Two-party setup for shared key cryptography for secure communication. Retrieved from *Introduction to Modern Cryptography* by Katz and Lindell

Introduction to Symmetric Cryptography

Formally, a private key encryption scheme is defined by a message space $\mathcal{M}$ (the set of messages supported by the scheme), along with three algorithms: A procedure for generation keys (**Gen**), a procedure for encrypting (**Enc**) and a procedure for Decrypting (**Dec**).

The algorithms are defined to have the following functionality:

1. The key-generation algorithm Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution.
2. The encryption algorithm Enc takes as input a key k and a message m and outputs a ciphertext c. We denote by $Enc_k(m)$ the encryption of the plaintext m using the key k.
3. The decryption algorithm Dec takes as input a key k and a ciphertext c and outputs a plaintext m. We denote the decryption of the ciphertext c using the key k by $Dec_k(c)$.

An encryption scheme must satisfy the following correctness requirement: for every key k output by **Gen** and every message $m \in \mathcal{M}$, it holds that

$$Dec_k(Enc_k(m)) = m$$

The set of all possible keys output by the key-generation algorithm is called the key space and is denoted by $\mathcal{K}$. Almost always, Gen simply chooses a uniform key from the key space.

Keys and Kerckhoffs' principle: The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.

Perfect Secrecy

Imagine an adversary who knows the probability distribution over $\mathcal{M}$. Faithful to Keys and Kerckhoff, the adversary also knows the encryption scheme in use. The adversary can eavesdrop and observe the ciphertext, yet has no way to observe the key.

Perfect Secrecy: An encryption scheme (**Gen**, **Enc**, **Dec**) with message space $\mathcal{M}$ is perfectly secret if for every probability distribution over $\mathcal{M}$, every message $m \in \mathcal{M}$, and every ciphertext $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$:

$$Pr[M = m|C = c] = Pr[M = m].$$

Perfect Secrecy: One Time pad

Discovered in 1882, the one time pad is a simple construction:
For a fixed length message of size *len*, with message space $\mathcal{M}$, key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$ all equal to $\{0,1\}^{len}$:

- **Gen**: Will choose a key according to uniform distribution
- **Enc**: given a message m and key k, outputs ciphertext $c := m \oplus k$
- **Dec**: given a ciphertext c and right key k, outputs $m := c \oplus k$

A properly constructed one time pad is perfectly secure! We're done right?

Perfect Secrecy: One Time pad (cont)

Except it's impractical. Imagine the key overhead for many large messages!
Along the same vein, it turns out that perfect secrecy isn't practical either.
Turns out a perfect encryption scheme with message space $\mathcal{M}$ and with key space $\mathcal{K}$ requires $|\mathcal{K}| \geq |\mathcal{M}|$

Computational Security

We can consider an encryption scheme to be good if it leaks only a tiny amount of information to an attacker with bounded computational power. Moreover, we must consider such a scheme to work for multiple messages to be effective. Thus, we must define constructions that possess nearly uniform distribution of probability for multiple distinct, incoming messages for a single key.

Computational Security: PRGs and Stream Ciphers

PRG: A pseudorandom generator G is an efficient, deterministic algorithm for transforming a short, uniform string called the seed into a longer, "uniform-looking" (or "pseudorandom") output string.

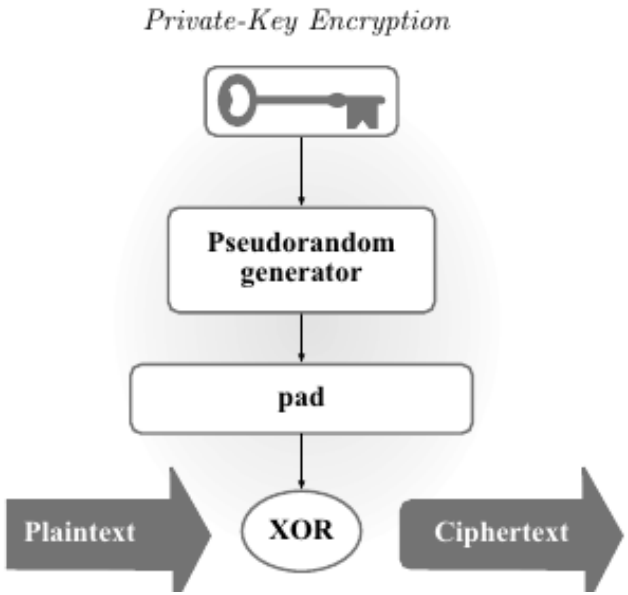
$$F : \{0,1\}^* \to \{0,1\}^*$$

PRF: A Pseudorandom Function follows the same idea of a PRG but more generalized: instead of taking a seed and producing a random string, we take some seed and some key k, and produce a pseudorandom function F_k. For our purposes F is length preserving with input, output and key length being equal

$$F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$$

PRP: A Pseudorandom permutation expands on the concept of a PRF, but with F essentially being a different permutation function. For a long enough block, a PRP is indistinguishable from a PRF.

As such, we can begin with the very simple cipher construction of a stream cipher, considering the key as the seed which generates a pad which we can simply xor with the plaintext.



However, a very basic stream cipher defined this way is not secure over multiple message encryptions.
As a matter of fact, If α is a (stateless) encryption scheme in which **Enc** is a deterministic function of the key and the message, then α cannot have indistinguishable multiple encryptions in the presence of an eavesdropper.

Computational Security: PRPs and Block Ciphers

To protect against chosen plaintext attacks, we must use a stronger notion of security. That is, we require the notion of security against multiple encryptions.
Block ciphers are designed to be secure instances of pseudorandom permutations (PRP) with some fixed key and block length.
We can use a similar notion of use fixed length blocks to create a stream cipher with a nonce or IV.
Now, despite this, block ciphers only operate on fixed lengths. We need a mechanism to process arbitrary lengths to be practical.

Computational Security: Modes of operation

For stream ciphers, we have:

1. Synchronized mode: For two party communications with a shared seed, sender and receiver are synchronized and know how much plaintext has been processed (enc/dec)
2. Unsynchronized mode: IV

Computational Security: Modes of operation

For block ciphers we have the following:

Block Cipher modes ECB

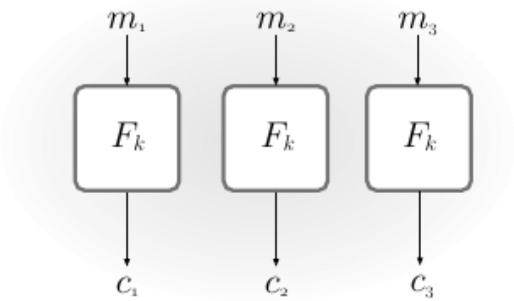
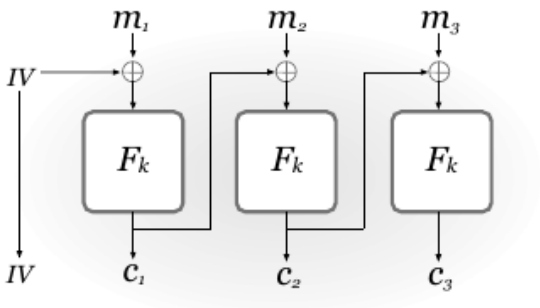
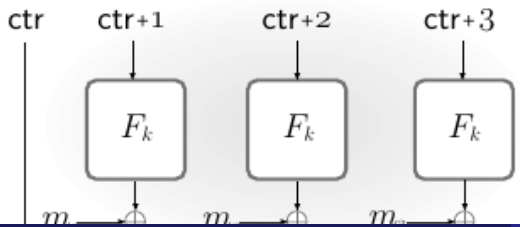


FIGURE 3.5: Electronic Code Book (ECB) mode.

CBC



CTR



Exercise time!!! Let's reverse engineer CTR mode!

Introduction to Hash functions

Cryptographic Hash Function: A way to map an arbitrary (non-zero) input string to a fixed length string, with the following properties for some hash function H:

- Must be collision resistant: It is infeasible for a PPT to find a collision in H. That is some pair x, x' such that $H(x) = H(x')$
- Preimage resistant: Infeasible for a PPT adversary to find, for some y, an x such that $H(x) = y$

Formally a hash function is defined as the following:
A hash function with output length *len* is a pair of polynomial time algorithms $(\mathbf{Gen}, H)$ that satisfy:

- $\mathbf{Gen}$ is probabilistic algorithm that takes in a security parameter 1^n and outputs a key k
- H takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in 0, 1^{(n)}$ (where n is the value of the security parameter implicit in s).

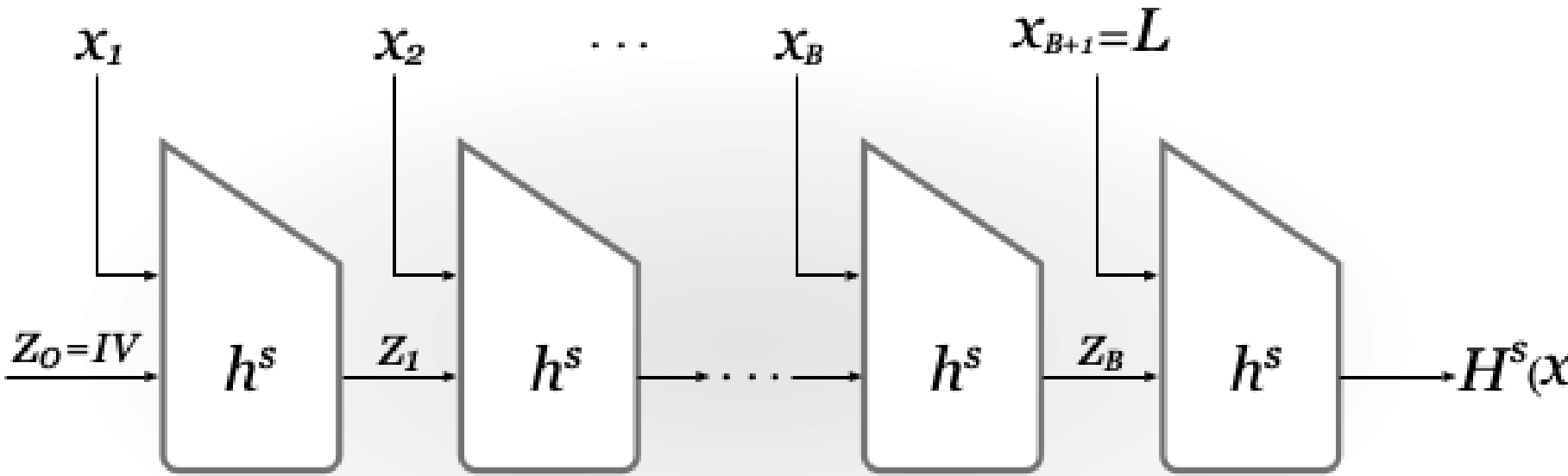
We can also call H in this case a compression function.

Hash functions

Despite defining hash functions above as keyed (and indeed, many keyed hash functions do exist), many of the hash functions used in practice are unkeyed, that is $H : \{0,1\}^* \rightarrow \{0,1\}^*$

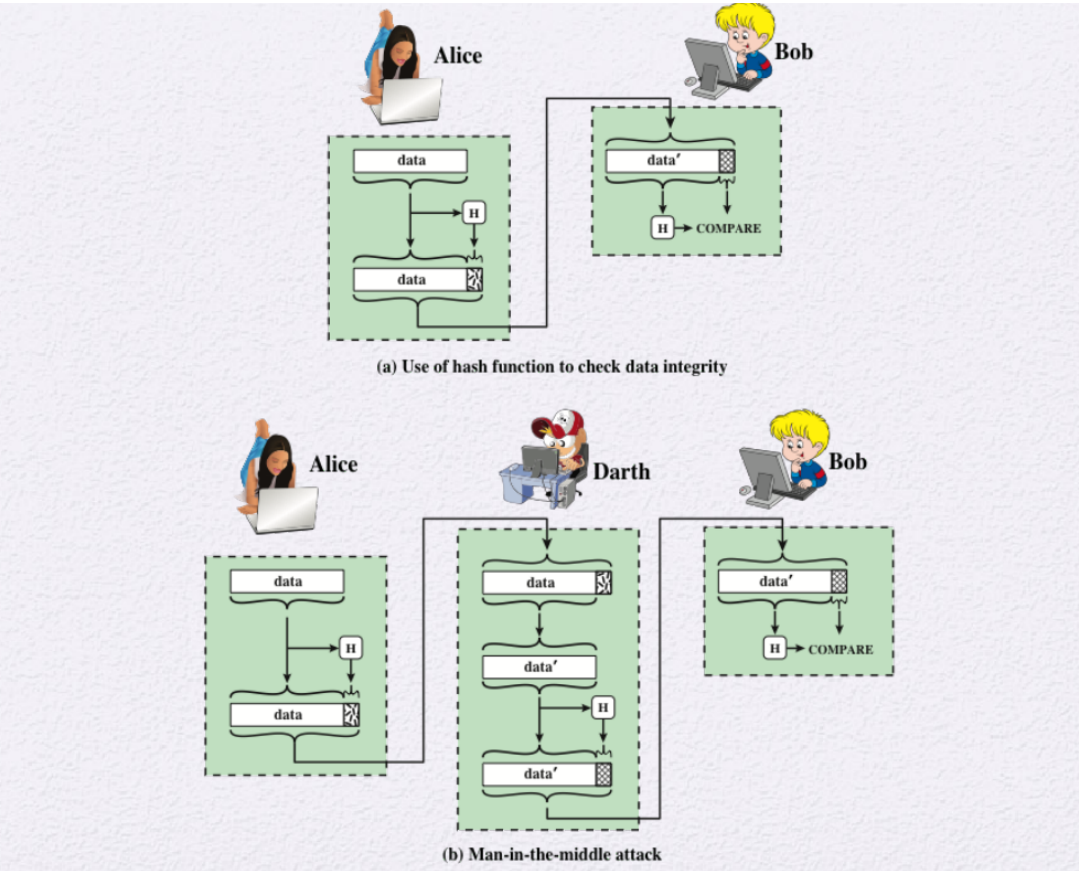
In practicality, hash functions are designed first by designing a collision-resistant compression function that handles inputs of a fixed length, then using domain extension to handle arbitrary-length inputs.

A common approach for domain extension is the Merkle-Damgård transform, shown below.



Hash functions

With hash functions, in particular, we're concerned with integrity. That is, we want to ensure authenticity of the data. However, hash functions alone are not enough to ensure integrity.



Exercise Time! Time to play with a very basic "blockchain"

Message Authentication

As we saw in the previous section, despite encryption providing secrecy and hash functions providing a deterministic but hard to invert pseudorandom string based on an input, neither of them solve the issue of providing *integrity*.

In many (dare I saw most?) applications, integrity is at least as important, if not more important than just secrecy. Think of authenticating a user, or a particular transaction. We would like the details of that transaction to be hidden, but more important than someone reading it, is someone

Message Authentication

Let's get formal: A message authentication scheme is a 3-tuple $(\mathbf{Gen}, \mathbf{Mac},$ with the following properties:

- **Gen** produces a symmetric key k (implicit security parameter 1^n, or takes an explicit security parameter).
- **Mac** accepts a message $m \in \{0,1\}^*$ and a secret key k and produces a a tag t. That is:

$$\mathbf{Mac} : k \times m \rightarrow t$$

- **Verify** accepts a message m, a key k and a tag t and returns a boolean value of whether the tag is matches. That is:

$$\mathbf{Verify} : k \times m \times t \rightarrow bool$$

Though Mac functions have been defined for fixed and arbitrary length inputs, we only care about the latter for practical reasons.

Message Authentication: CBC-Mac

As a first taste of how we could think of creating a secure Mac function, we know we require a pseudorandom tag for an arbitrary message m, and we have at our disposal *PRF*s as well as cipher modes of operations. What can we do?
Enter CBC-Mac: Let F be a *PRF* that accepts some key k and messages m of length l of some block length n:

- Split m into $m_1, m_2, ..., m_(l/n)$ (you may need to pad m)
- Set $t_0 = 0^n$. For $i = 1$ to (l/n) : $t_i = F(t_{i-1} \oplus m_i)$
- return $t_{l/n}$

HMac

HMac is the industry golden standard for Message authentication at the moment. Despite this, it is a fairly simple construction (using Merkle-Damgård transforms for arbitrary length messages). For some secure hash function H, key k and message m:

$$HMAC(k, m) = H\Big((k \oplus opad)||H\big((k \oplus ipad)||m\big)\Big)$$

Where *opad* and *ipad* are fixed paddings to derive different keys for the outer and inner hash computations, which provide additional security under the assumption that H is weak collision resistant (This proved useful in practice!)

Authenticated Encryption

Coming back to encryption, now that we've defined message authentication codes, we can combine the notion of a message authentication code with encryption to form authenticated encryption.
However, when should we apply it? There are currently three notions:

- Encrypt-then-Mac (current preferred in industry, i.e AES-GCM or XSalsa20Poly1305)
- Encrypt-and-Mac
- Mac-then-encrypt

Exercise time! Did anyone say JWT?

Introduction to number theory

At the most basic level, Number Theory is the study of the integers. We denote the set of integers by the symbol $\mathbb{Z}$.

Divisibility: Let a and b be integers with $b \neq 0$. We say "b divides a", "a is divisible by b" or $b \mid a$ if there exists an integer c such that $a = b * c$

Let $a, b, c \in \mathbb{Z}$:

- if $a \mid b$ and $b \mid c$ then $a \mid c$
- if $b \mid a$ and $a \mid b$ then $a = \pm b$
- if $a \mid$ b and $a \mid c$ then $a \mid (b + c)$ and $a \mid (b - c)$

Introduction to number theory

A common divisor of two integers is one that divides both of them. A greatest common divisor, or *gcd*, is the largest number such that it divides both of them. The greatest common divisor for some two arbitrary integers a and b is $gcd(a, b)$.

Division with remainder: Let a and b be positive integers. we say a divided by b has a quotient q and remainder r if:

$$a = b * q + r$$

(with q and r unique).
Of course if $b \mid a$, then r $= 0$.

Euclidean Algorithm

Algorithm to find the gcd of two positive integers (explained on board).
Extended Euclidean algorithm: Let a and b two positive integers, then

$$au + bv = gcd(a, b)$$

Always has a solution. A special case arises when $gcd(a, b) = 1$. In that case we say a and b are *relatively prime* or *coprime*.

Lightning overview: Modular Arithmetic

Definition: Let $m \geq 1$ be an integer. We say that the integers a and b are congruent modulo m if their difference $a - b$ is divisible by m. We write

$$a \equiv b \pmod{m}$$

To indicate that congruence. m is called the modulus.
Properties of modular arithmetic:

- If $a1 \equiv a2 \pmod{m}$ and $b1 \equiv b2 \pmod{m}$ then $a1 \pm b1 \equiv a2 \pm b2 \pmod{m}$
- $a * b \equiv 1 \pmod{m} \iff gcd(a, m) = 1$. We also call b in this scenario the multiplicative inverse of a modulo m

Lightning overview: Groups/Rings/Fields

Definition: A group consists of a set $\mathcal{G}$ and a rule, which we denote by $\star$, for combining two elements $a, b \in \mathcal{G}$ to obtain an element $a \star b \in \mathcal{G}$. The composition operation $\star$ is required to have the following three properties:

- Identity law: $\{\exists! \epsilon \in \mathcal{G} \, \forall a \in \mathcal{G} : \epsilon \star a = a \star \epsilon = a\}$
- Associative law: $\{\forall a, b, c \in \mathcal{G} : a \star (b \star c) = (a \star b) \star c\}$
- Inverse law: $\{\forall a \in \mathcal{G} \, \exists! a^{-1} \in \mathcal{G} : a * a^{-1} = \epsilon\}$

If for some group $\mathcal{G}$ the $\star$ operator is commutative, we call that a commutative or abelian group.

Lightning overview: Groups/Rings/Fields

Definition: A ring is a set $\mathcal{R}$ that has two operations $\{+, \star\}$ (addition and multiplication) with the following properties:

- $+$ forms an abelian group with an identity element denoted as 0.
- $\star$ is a monoid under multiplication.
- $\star$ is distributive with respect to $+$. That is:

$$a \star (b + c) = (a \star b) + (a \star c) (\text{Left distributivity})$$

$$(b + c) \star a = (b \star a) + (c \star a) (\text{Right distributivity})$$

If $\star$ is commutative (that is elements of $\mathcal{G}$ under $\star$ form a commutative monoid), then we can call $\mathcal{R}$ a commutative ring.
If a commutative ring $\mathcal{R}$ also satisfies the inverse law for $\star$, we call it a *Field*. Fields may be finite or infinite, but in cryptography, we mostly concern ourselves with finite fields.

Lightning overview: Primes, Finite Fields

Fundamental theorem of arithmetic: Every integer ≥ 2 has a unique prime factorization.
We denote for some integer m the set $\mathbb{Z}/m\mathbb{Z}$ as the ring of integers modulo Z.
For some prime p, the set $\mathbb{Z}/p\mathbb{Z}$ forms a field, more commonly denoted as $\mathbb{F}_p$. Since, as we saw for every prime and some $a \in (1..p-1)$, $gcd(a, p) = 1$, every element of a field has a multiplicative inverse.

Fermat's little theorem: Let p be a prime number and let a be any integer. Then:

- $a^{p-1} \equiv 1 \pmod{p}$ if $p \nmid a$
- $a^{p-1} \equiv 0 \pmod{p}$ if $p \mid a$

Lightning overview: Primes, Finite Fields

Primitive Root Theorem:. Let p be a prime number. Then there exists an element $g \in \mathbb{F}_p$ whose powers give every element of $\mathbb{F}_p$.
Elements with this property are called primitive roots of $\mathbb{F}_p$ or generators. They are the elements of $\mathbb{F}_p$ having order $p - 1$.

Asymmetric Cryptography: Brief history

Back in 1976, Whitfield Diffie and Martin Hellman published their groundbreaking paper "New Directions in Cryptography". In it, they described the first public key cryptosystem and a fundamental aspect of PKC: One way functions.

A one-way function is an invertible function that is easy to compute, but whose inverse is difficult to compute. We have already seen one way functions with respect to hash functions, but we did not define them as such.

However, despite their fundamental important in PKC, it is only speculated that one-way functions exist. Their proliferance in PKC and prominence in practice point to indications that they should exist, however one-way functions are inherently tied to complexity theory, to the point where a proof of their existence would solve the $\mathcal{P} = \mathcal{NP}$ problem

Enter the DLP

To understand the Diffie-Helman key exchange, we must first understand the fundamental problem that DH is based upon: The discrete logarithm problem.

Definition: Let $\mathcal{G}$ be a group whose binary operator we denote with $\star$. The Discrete Logarithm Problem for $\mathcal{G}$ is to determine any two given elements $(g, h) \in \mathcal{G}$ such that:

$$g \star g \star g \star ... \star g = g^x = h$$

For a finite field $\mathbb{F}_p$, this translates to, for some generator g:

$$g^x = h \pmod{p}$$

We can see that despite the DLP being a relatively simple problem, it has an incredibly powerful implication: Solving the for an arbitrary exponent x by brute force, for a large field $\mathbb{F}_p$ can be hard.

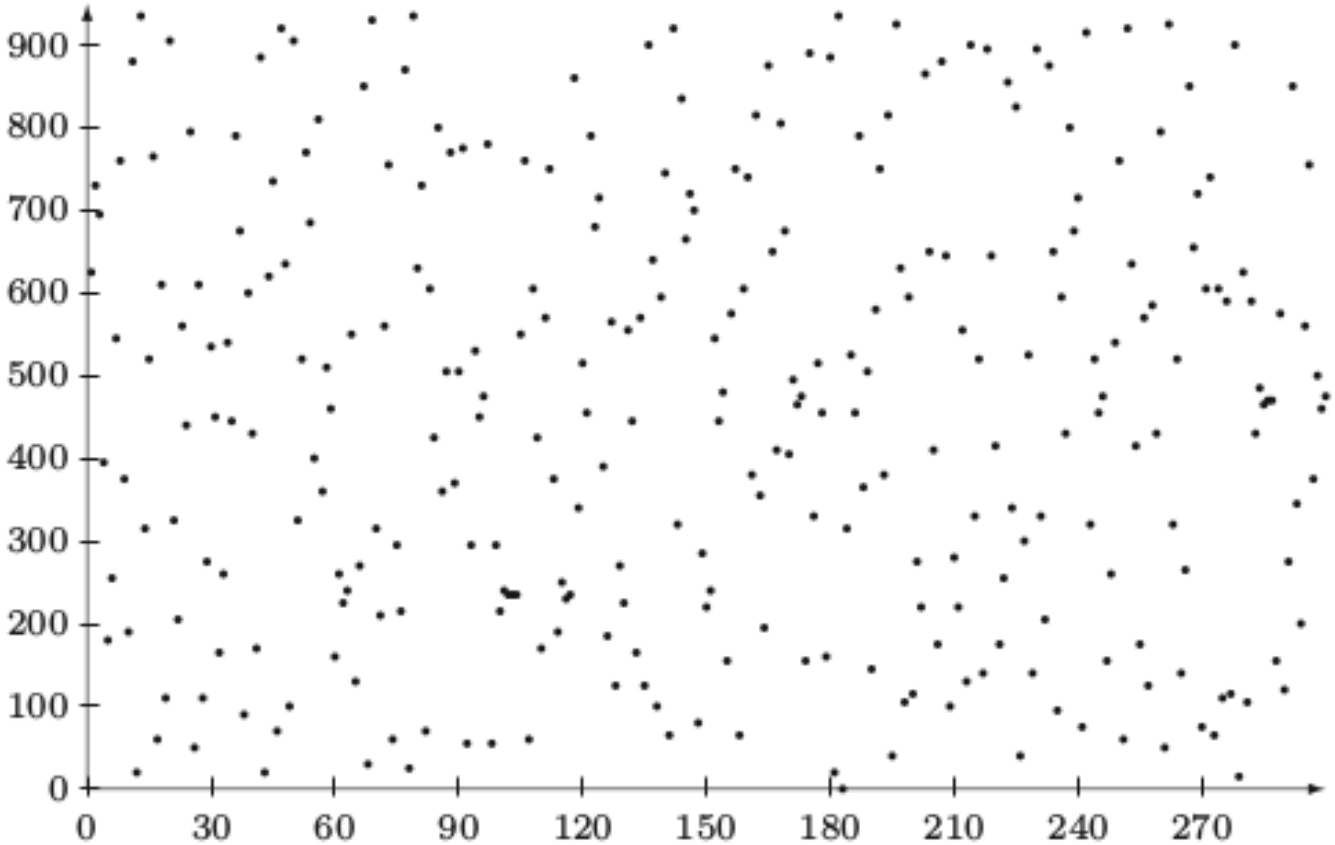


Figure 2.2: Powers $627^i \bmod 941$ for $i = 1, 2, 3, \ldots$

Diffie-Hellman

The Diffie-Hellman key exchange solves the problem is establishing a key between two parties under an insecure channel.

Public parameter creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in $\mathbb{F}_p^*$.	
Private computations	
Alice	**Bob**
Choose a secret integer a.	Choose a secret integer b.
Compute $A \equiv g^a \pmod{p}$.	Compute $B \equiv g^b \pmod{p}$.
Public exchange of values	
Alice sends A to Bob $\longrightarrow$	A
B $\longleftarrow$	Bob sends B to Alice
Further private computations	
Alice	**Bob**
Compute the number $B^a \pmod{p}$.	Compute the number $A^b \pmod{p}$.
The shared secret value is $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	

Difficulty of the DLP + Attacks

How hard is the DLP? A basic intiution tells us that for a generator $g \in \mathbb{F}_p$ of order n, solving $g^x = h \pmod{p}$ only takes n multiplications. However, in practical applications we often use large primes (1024, 2048 and more bits), thus solving for 2^{1024} by simple modular arithmetic and iteration becomes longer than any of us will live.

Thought Exercise: How to beat Diffie hellman?

RSA and Integer Factorization

Euler's Formula for pq: Let p and q be distinct primes and let $g = gcd(p-1, q-1)$. Then $a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$.

Let p and q be distinct primes and let $e \geq 1$ satisfy

$$gcd(e, (p-1)(q-1)) = 1$$

. We know e has an inverse modulo, thus

$$de \equiv 1 \pmod{(p-1)(q-1)}$$

.
Then the congruence:

$$x^e \equiv c \pmod{pq}$$

has the unique solution $x = c^d$

RSA and Integer Factorization

RSA: An asymmetric cryptosystem for exchanging sensitive information over a communication line, we can use the constructions we just saw to send and receive an encrypted message.

Bob	Alice
Key creation	
Choose secret primes p and q. Choose encryption exponent e with $\gcd(e, (p-1)(q-1)) = 1$. Publish $N = pq$ and e.	
Encryption	
	Choose plaintext m. Use Bob's public key (N, e) to compute $c \equiv m^e \pmod{N}$. Send ciphertext c to Bob.
Decryption	
Compute d satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$. Compute $m' \equiv c^d \pmod{N}$. Then m' equals the plaintext m.	

RSA

You may have noticed: The Strength of RSA is very clearly tied to the difficulty of integer factorization. If we can factor N into p and q, the entire scheme breaks.
True to this, RSA inspired advanced in number theory, in particular to find more efficient methods of integer factorization. We will see one such attack here

Index Calculus

$$X^2 - Y^2 = (X - Y)(X + Y)$$

Is one of the most powerful yet simple methods of factorization known today.

The following is a summarized version of integer factorization:

1. **Relation Building**: Find many integers $a_1, a_2, a_3, \ldots, a_r$ with the property that the quantity $c_i \equiv a_i^2 \pmod{N}$ factors as a product of small primes.
2. **Elimination**: Take a product $c_{i_1} c_{i_2} \cdots c_{i_s}$ of some of the c_i's so that every prime appearing in the product appears to an even power. Then $c_{i_1} c_{i_2} \cdots c_{i_s} = b^2$ is a perfect square.
3. **GCD Computation**: Let $a = a_{i_1} a_{i_2} \cdots a_{i_s}$ and compute the greatest common divisor $d = \gcd(N, a - b)$. Since

$$a^2 = (a_{i_1} a_{i_2} \cdots a_{i_s})^2 \equiv a_{i_1}^2 a_{i_2}^2 \cdots a_{i_s}^2 \equiv c_{i_1} c_{i_2} \cdots c_{i_s} \equiv b^2 \pmod{N},$$

there is a reasonable chance that d is a nontrivial factor of N.

Table 3.4: A three step factorization procedure

Exercise time! Let's factorize RSA a tad.

Digital Signatures

Digital Signatures are what we would like to be the equivalent of our pen and paper signatures, but digitally.
A digital signature scheme is a 3-tuple $(\mathbf{Gen}, \mathbf{Sign}, \mathbf{Verify})$ with the following properties:

- **Gen** produces a key pair (pk, sk).
- **Sign** accepts a message m and a secret key sk and produces a signature σ. That is:

$$\mathbf{Sign} : k_{private} \times m \rightarrow \sigma$$

- **Verify** accepts a message m, a public key pk and a signature m and returns a boolean value of whether the signature is correct. That is:

$$\mathbf{Verify} : k_{public} \times m \times \sigma \rightarrow bool$$

RSA Digital Signatures

One of the simplest schemes we can think of for digital signatures is simply using the RSA encrypted value. Thus, we have RSA digital signatures.

Samantha	Victor
Key creation	
Choose secret primes p and q. Choose verification exponent e with $\gcd(e, (p-1)(q-1)) = 1$. Publish $N = pq$ and e.	
Signing	
Compute d satisfying $de \equiv 1 \pmod{(p-1)(q-1)}$. Sign document D by computing $S \equiv D^d \pmod{N}$.	
Verification	
	Compute $S^e \bmod N$ and verify that it is equal to D.

However, this doesn't scale very well, and cannot handle messages longer than your N value (Why?)

Hash-N'-Sign

We can augment our Digital signature scheme by using a strong cryptographic hash function.
We can then redefine it formally by augmenting our previous definition with $(\mathbf{Gen}, H, \mathbf{Sign}, \mathbf{Verify})$, that carries following properties:

- **Gen** produces a key pair (pk, sk), as before.
- **Sign** instead of operating on m, we feed the same signing function $H(m)$ instead
- **Verify** instead of verifying (m, σ), we now verify $(H(m), \sigma)$

EXERCISE TIME!!!

Elliptic Curves: Intro

An Elliptic Curve is a set of solutions to *Weierstrass equations*:

$$Y^2 = X^3 + AX + B$$

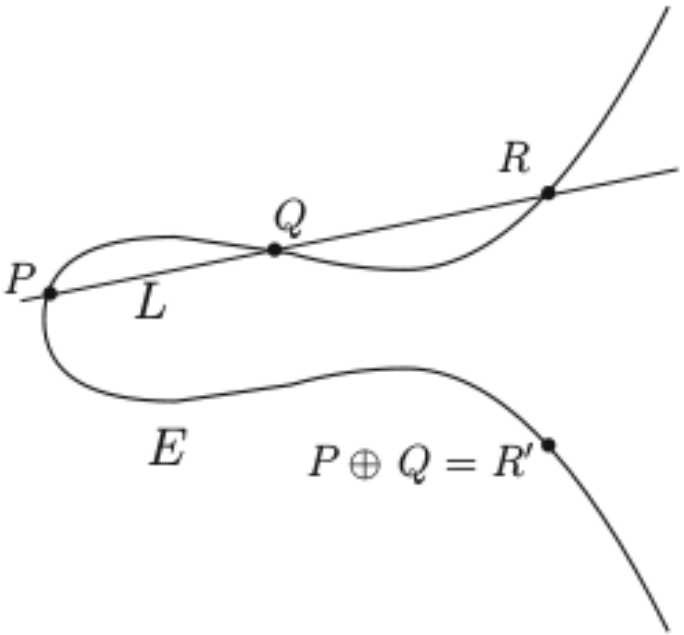
Said equation also comes with an additional constraint, called the *discriminant*, where the above constants A and B satisfy:

$$4A^3 + 27B^3 \neq 0$$

A beautiful property of Elliptic curves is that they behave in ways like to the algebraic laws we saw before. You can take two points on an elliptive curve an "add" them together(Addition of two points is somewhat different than a traditional notion of addition)

Elliptic Curves: Algebra

To add two points (P, Q) on an elliptic curve, we draw a line L through them. L will intersect the curve at 3 points (Distinctness of the points depends on whether $P = Q$). We denote elliptic curve addition of the two points as $P \oplus Q$



Elliptic Curves: Algebra

Some of you may have noticed that for a point $P = (a, b)$, the addition method breaks apart when we consider inverting the y axis, that is $P' = (a, -b)$. For that, we create an extra point, denoted by $\mathcal{O}$ which lives at infinity. We then set $P \oplus P' = \mathcal{O}$ and call it a day.

The nice part about this special point, is that it can act as an algebraic identity. That is, $P + \mathcal{O} = \mathcal{O} + P = P$

Multiplication is easy for elliptic curves. While not being traditionally multiplication (so not distributive against the coordinates over some constant), multiplication is simply defined as repeated addition. That is:

$$nP = P + P + ... + P \, (\text{n times})$$

If we consider Elliptic Curve addition happens to be commutative and associative, then we can form an Abelian group!

Elliptic Curves: Algebra

With groups we are close, but from what we saw earlier, what we would really love is to have a field. What if?
Well, it turns out we can. An Elliptic curve over a Finite field $\mathbb{F}_p$, denoted as $E(\mathbb{F}_p)$ is an equation of the form:

$E : Y^2 = X^3 + AX + B$ with $A, B \in \mathbb{F}_p$ satisfying the discriminant $4A^3 + 27B^2 \neq 0$
The set of points on E must satisfy the equation modulo p

ECDLP

It turns out we can define the DLP over Elliptic curves as well. That is:

$$Q = nP, Q, P \in E(\mathbb{F}_p)$$

Why do we care?

- The ECDLP does not have an index calculus attack. At the moment, the fastest attack on the ECDLP has $O(\sqrt{p})$ complexity. This is good!
- The ECDLP is not subject to the pitfalls of integer factorization. Thus, we can use smaller keys (RSA Keys can be 2048 bits or more! EC Keys come in the same sizes AES keys do)

Let us derive Elliptic curve Diffie-Hellman!

Thank you!!