A (Slightly Deranged) Survey of (Simple) Cryptography

“We kill people based on metadata.” –Michael Hayden[1]

Ananth Venkatesh

Massachusetts Institute of Technology (MIT)

May 7, 2025

[1] Attribution: (Ferran 2014) (famously also believes the Fourth Amendment is not a real thing)

Outline

Security by obfuscation is security by delusion.

(cryptography is the study of the mathematical theory of *provably secure systems*)

1. Classical Cryptography

Figure 1: The NSA is, and has remained, enemy number one; but American corporations are not far behind

1.1 Hashing

The foundations of cryptography are built on one-way functions, called **hashes**.

1.1 Hashing

The foundations of cryptography are built on one-way functions, called **hashes**.

Definition

Hashes are formally defined as injective functions $f : X \to Y$, where X is some data we want to hash and Y is a fixed-length bitstring. In reality, X can be a set with infinite cardinality, so the fixed length of Y means it is theoretically impossible to construct such an injective function.

1.1 Hashing

Ideal hashes satisfy the following properties:

1.

2.

3.

1.1 Hashing

Ideal hashes satisfy the following properties:

1. Few collisions $(x, y) \mid f(x) = f(y)$ and collision resistance (hard to find (x, y))
2.
3.

1.1 Hashing

Ideal hashes satisfy the following properties:

1. Few collisions $(x, y) \mid f(x) = f(y)$ and collision resistance (hard to find (x, y))
2. f is easy to compute and well-defined, but f^{-1} is exceedingly difficult to compute
3.

1.1 Hashing

Ideal hashes satisfy the following properties:

1. Few collisions $(x, y) \mid f(x) = f(y)$ and collision resistance (hard to find (x, y))
2. f is easy to compute and well-defined, but f^{-1} is exceedingly difficult to compute
3. Uniform distribution of output and maximum entropy (the smallest discrete change in input should yield the maximum possible expected change in output)

1.1 Hashing

1.1.1 Entropy

In the case of an input bitstring, changing one bit in $x \in X$ should flip, on average, half the bits in $y \in Y$.

1.1 Hashing

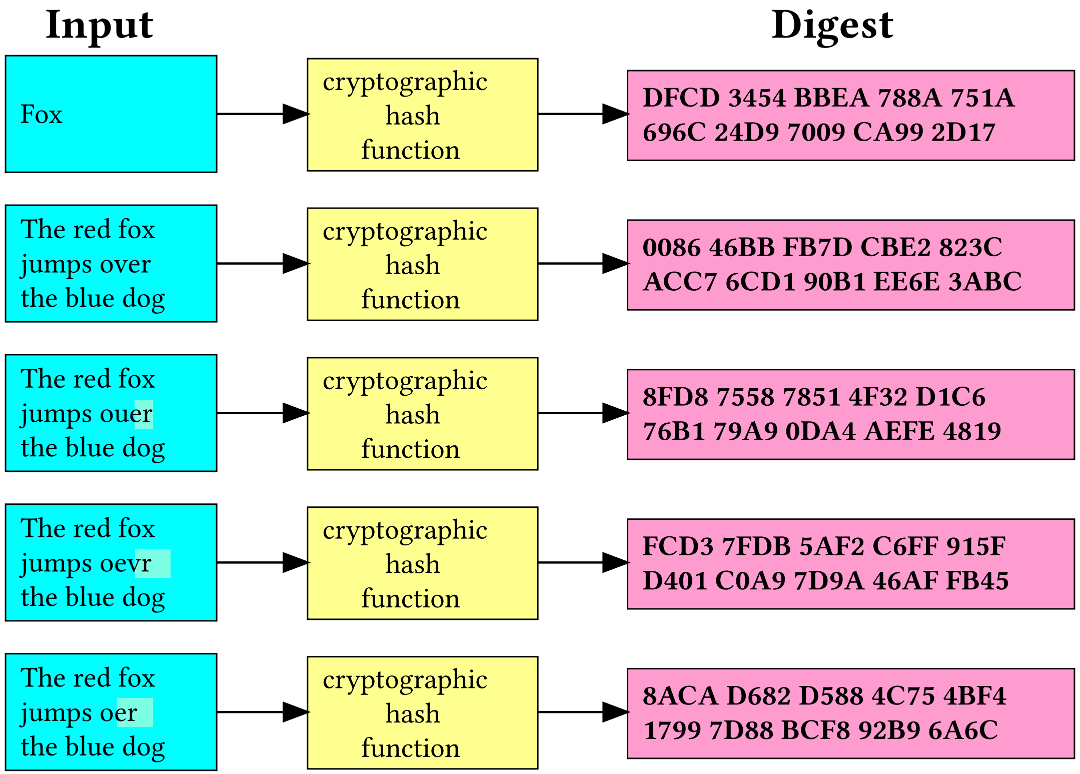


Figure 1: By User:Jorge Stolfi based on Image:Hash_function.svg by Helix84 - Original work for Wikipedia, Public Domain, commons.wikimedia.org (Helix84 2008)

1.1 Hashing

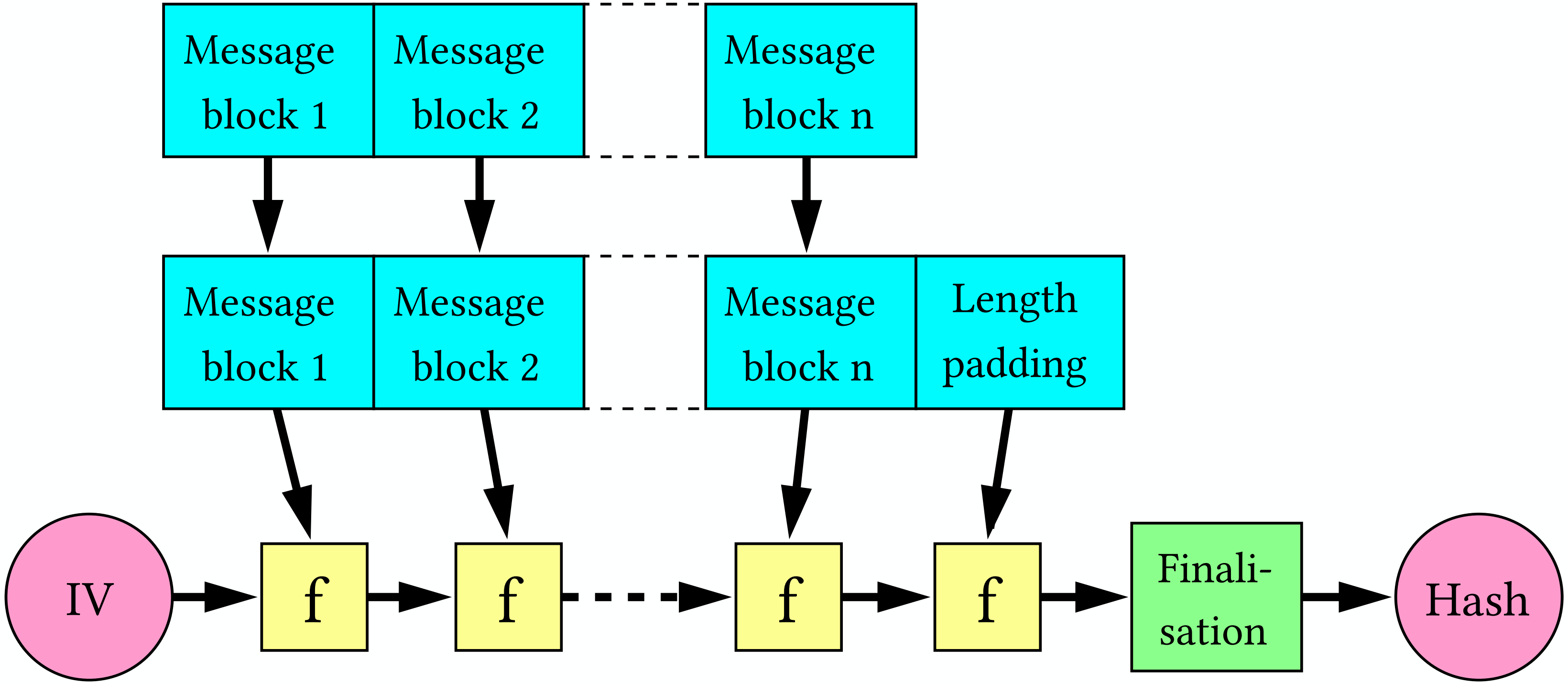


Figure 2: Process message of arbitrary size in chunks using a compression algorithm in series
By Davidgothberg - Own work, Public Domain, commons.wikimedia.org (Davidgothberg 2007)

1.1 Hashing

- Creating such a function (that is cryptographically secure) is notoriously difficult

1.1 Hashing

- Creating such a function (that is cryptographically secure) is notoriously difficult
- Most involve obscure bit transformations that read like dark magic

1.1 Hashing

- Creating such a function (that is cryptographically secure) is notoriously difficult
- Most involve obscure bit transformations that read like dark magic
- DIY hash functions (and DIY anything in cryptography) is frowned upon by experts due to the necessity for rigorous testing before something is deemed “secure”

1.1 Hashing

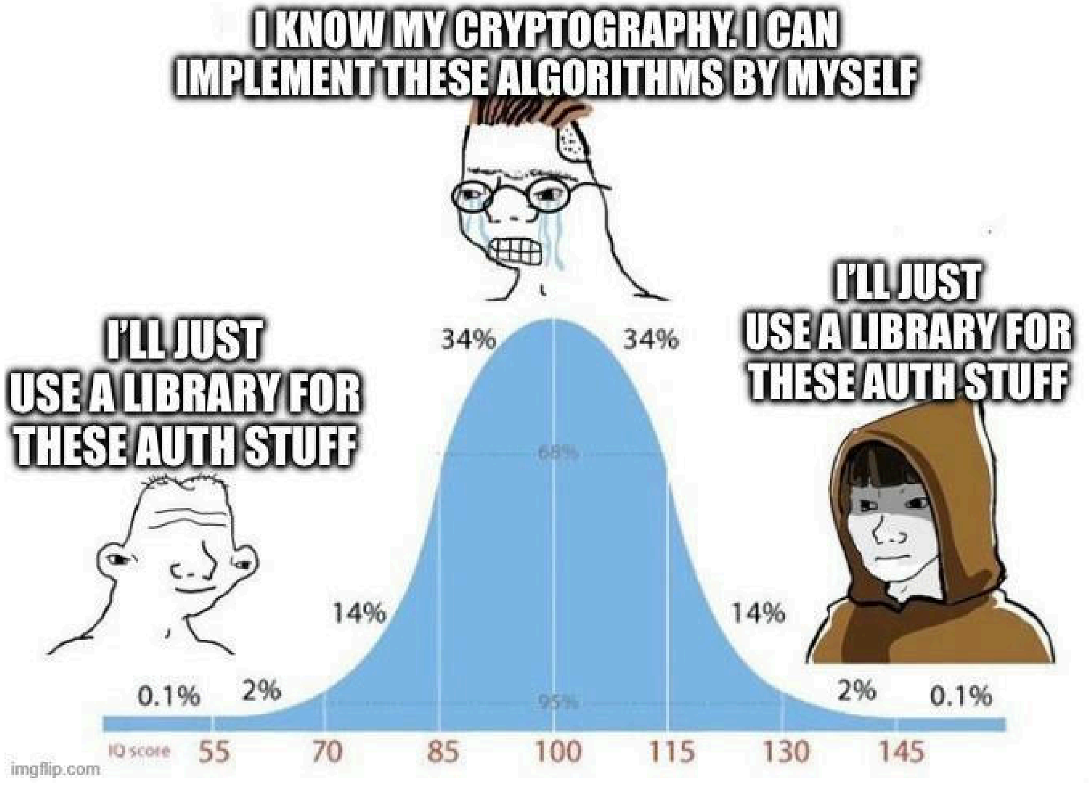


Figure 3: Typical cryptography learning curve

1.1.2 Applications

- Hash passwords in a database (need for **cryptographic salt**)

1.1.3 Applications

- Hash passwords in a database (need for **cryptographic salt**)
- Pseudorandom number generation (hash predictable input, get unpredictable output)

1.1 Hashing

1.1.4 Applications

- Hash passwords in a database (need for **cryptographic salt**)
- Pseudorandom number generation (hash predictable input, get unpredictable output)
- Verifying message integrity

"¡WhatsApp imperialismo tecnológico está atacando a Venezuela! … Soy libre de WhatsApp. Estoy en paz."
—Nicolás Maduro

1.2 Symmetric Encryption

- Want a way to encrypt data, readable by only one person

1.2 Symmetric Encryption

- Want a way to encrypt data, readable by only one person
- Use an algorithm that takes a secret and data, and then manipulates the data so it can only be retrieved if the secret is known

1.2 Symmetric Encryption

- Want a way to encrypt data, readable by only one person
- Use an algorithm that takes a secret and data, and then manipulates the data so it can only be retrieved if the secret is known
- Many algorithms to do this (trivial example is the one-time pad, but better, more efficient implmenetations exist)

1.2 Symmetric Encryption

Figure 1: The horrors of modern cryptography

1.3 Signatures

We want a way to verify the authenticity of a message.

1.3 Signatures

We want a way to verify the authenticity of a message.

How can I be sure that Hegseth really sent that message about the change in war plans?

1.3 Signatures

We want a way to verify the authenticity of a message.

How can I be sure that Hegseth really sent that message about the change in war plans?

To do this, we need to introduce the idea of a **key**.

1.3 Signatures

Definition

Keys are fixed-length random bitstrings that are either kept secret (private) or exposed (public). They act as encryption and decryption mechanisms.

1.3 Signatures

Definition

Keys are fixed-length random bitstrings that are either kept secret (private) or exposed (public). They act as encryption and decryption mechanisms.

- Generate a **private** key S for signing $f : M \to S \to$ Signature

1.3 Signatures

Definition

Keys are fixed-length random bitstrings that are either kept secret (private) or exposed (public). They act as encryption and decryption mechanisms.

- Generate a **private** key S for signing $f : M \to S \to \text{Signature}$
- Generate a corresponding **public** key V for verifying a signature $g : M \to V \to \text{Signature} \to \text{Bool}$

1.3 Signatures

Definition

Keys are fixed-length random bitstrings that are either kept secret (private) or exposed (public). They act as encryption and decryption mechanisms.

- Generate a **private** key S for signing $f : M \to S \to \text{Signature}$
- Generate a corresponding **public** key V for verifying a signature $g : M \to V \to \text{Signature} \to \text{Bool}$
- g verifies that a message M was signed with S

1.3 Signatures

Definition

Keys are fixed-length random bitstrings that are either kept secret (private) or exposed (public). They act as encryption and decryption mechanisms.

- Generate a **private** key S for signing $f : M \to S \to \text{Signature}$
- Generate a corresponding **public** key V for verifying a signature $g : M \to V \to \text{Signature} \to \text{Bool}$
- g verifies that a message M was signed with S
- Since S is private (known only to the sender), we can be sure the message is authentic if g returns true

1.3 Signatures

Figure 1: Hegseth is maxxing out his team's OPSEC by signing his messages so random journalists can be sure of their authenticity

1.4 Diffie–Hellman

- We have a lot of very good, very strong symmetric encryption algorithms

1.4 Diffie–Hellman

- We have a lot of very good, very strong symmetric encryption algorithms
- These are good if we want to send a message to ourselves, or if we are communicating with someone else but have a shared secret before we start

1.4 Diffie–Hellman

- We have a lot of very good, very strong symmetric encryption algorithms
- These are good if we want to send a message to ourselves, or if we are communicating with someone else but have a shared secret before we start
- If we are communicatinig over an insecure channel with no pre-arranged secret (connecting to servers over the internet, e.g.), everything we've built so far seems to break down

What if we could turn any symmetric encryption algorithm into an *asymmetric* one?

1.4 Diffie–Hellman

- The central idea of Diffie-Hellman is that we can arrive on a shared secret even if we are communicating over an insecure channel

1.4 Diffie–Hellman

- The central idea of Diffie-Hellman is that we can arrive on a shared secret even if we are communicating over an insecure channel
- We can do this because *it doesn't matter what the secret we arrive on is*, so long as it's (a) random and (b) secret (known only to both of us)

1.4 Diffie–Hellman

- The central idea of Diffie-Hellman is that we can arrive on a shared secret even if we are communicating over an insecure channel
- We can do this because *it doesn't matter what the secret we arrive on is*, so long as it's (a) random and (b) secret (known only to both of us)
- We circumvent the difficulty of communicating information over an insecure channel, because we are not communicating a coherent, known message but rather following an algorithm to create a shared random secret

1.4 Diffie–Hellman

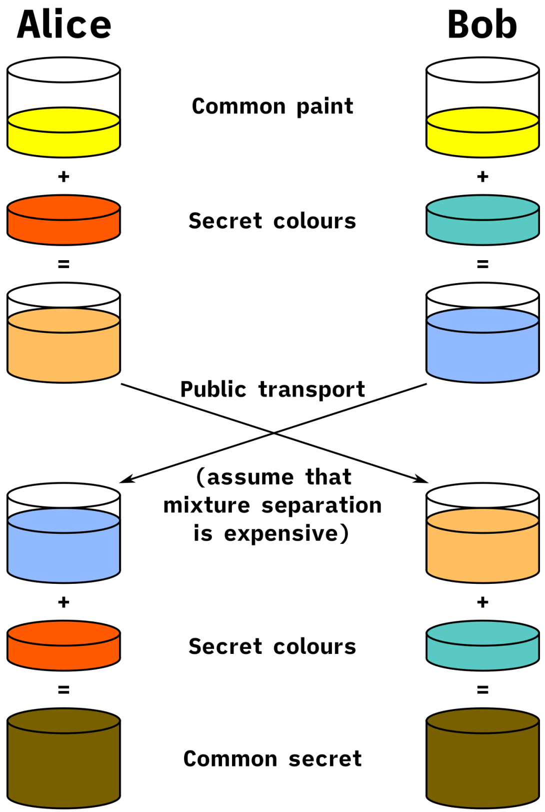


Figure 1: By Original schema: A.J. Vinck, University of Duisburg-EssenSVG version: Flugaal - A.J. Han Vinck, Introduction to public key cryptography, p. 16, Public Domain, commons.wikimedia.org (Helix84 2008)

1.4 Diffie–Hellman

- Replace paint with **keys** (red and green paint are *secret* keys, yellow is *public*)
- Replace addition with a one-way (hash-like) commutative binary operation

1.4 Diffie–Hellman

Implementation

1. Choose a cyclic group G of order n (where n is very, very large)
2. Choose a generating element $g \in G$ where g is the **public key**
3. Alice chooses a **secret key** a and Bob chooses a **secret key** b
4. Alice sends g^a and Bob sends g^b publicly
5. Alice computes $\left(g^b\right)^a$ and Bob computes $\left(g^a\right)^b$, the **shared secret**

1.4 Diffie–Hellman

We can verify that some basic properties hold:

1.4 Diffie–Hellman

We can verify that some basic properties hold:

- Multiplication of elements in a cyclic group is commutative ($(g^a)^b = g^{ab} = \left(g^b\right)^a$), so the computed sharerd secret is the same for Alice and Bob

1.4 Diffie–Hellman

We can verify that some basic properties hold:

- Multiplication of elements in a cyclic group is commutative ($(g^a)^b = g^{ab} = \left(g^b\right)^a$), so the computed sharerd secret is the same for Alice and Bob
- We can choose a group that gives rise to the "one-way" criterion: consider modular exponentiation of g ($G = \{g^k \bmod p \mid k \in \mathbb{Z}\}$ for some large prime p)

1.4 Diffie–Hellman

- Commutativity of this group comes from the fact that $(ab \bmod p) = (a \bmod p)(b \bmod p) \bmod p$

1.4 Diffie–Hellman

- Commutativity of this group comes from the fact that $(ab \bmod p) = (a \bmod p)(b \bmod p) \bmod p$
- Finding $g \mid g^k = h \bmod p$ given h, k is very hard for well-chosen p (this is known as the "discrete logarithm problem")

1.4 Diffie–Hellman

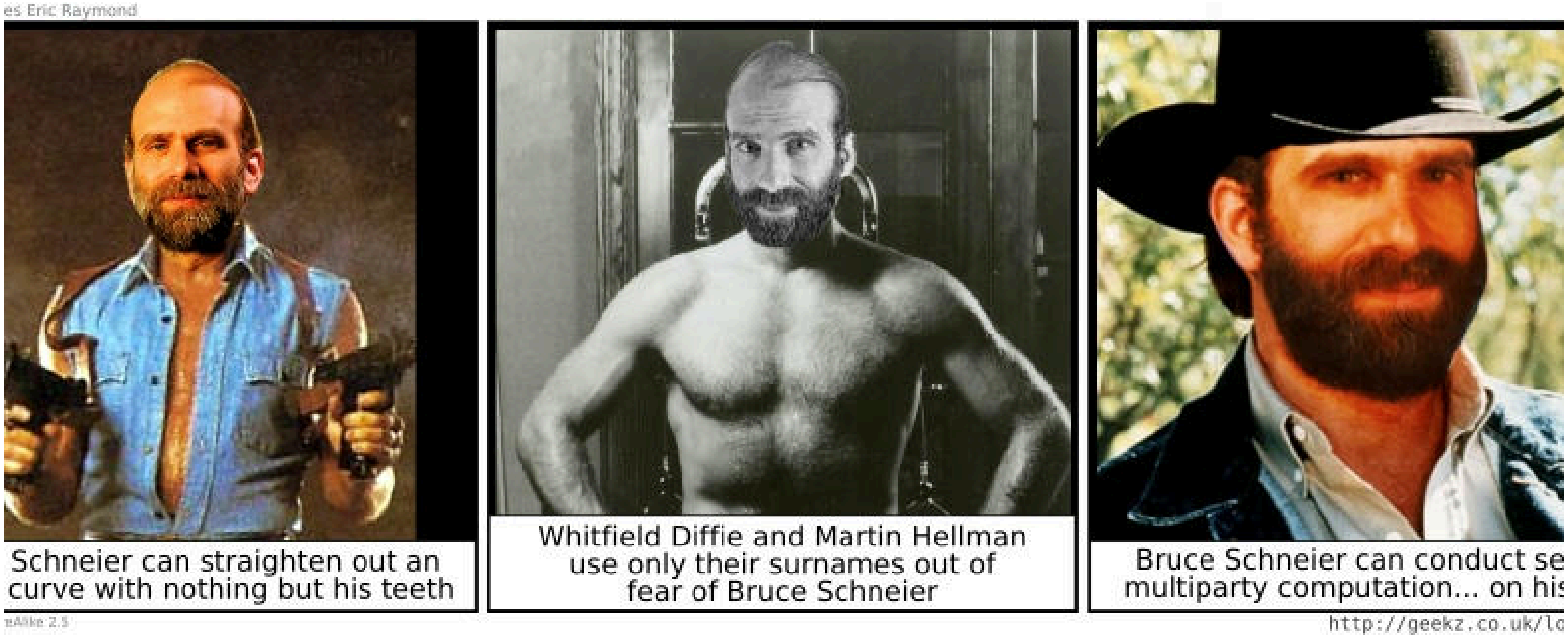


Figure 1: Friendly rivalry among prominent cryptographers are occassionally settled through unencrypted communication

1.5 Elliptic Curves

- Cool mathematical objects; they look like $y^2 = x^3 + ax + b$

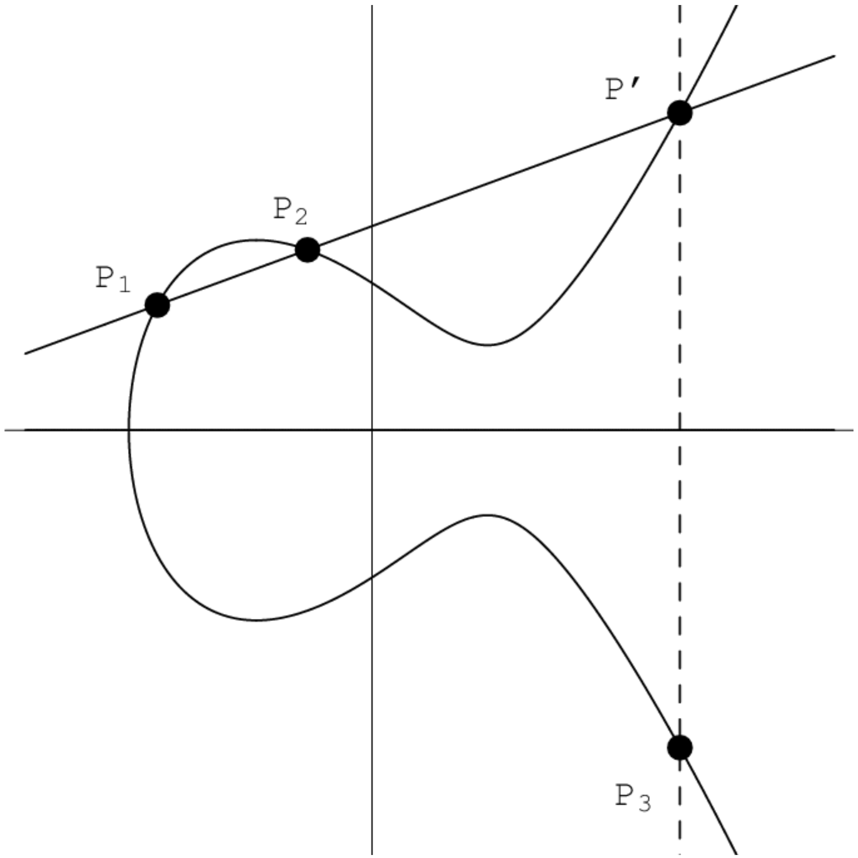
1.5 Elliptic Curves

- Cool mathematical objects; they look like $y^2 = x^3 + ax + b$
- "Addition" of points on an elliptic curve provide a much harder analogue of the discrete logarithm problem (which means smaller key sizes)

1.5 Elliptic Curves

- Cool mathematical objects; they look like $y^2 = x^3 + ax + b$
- "Addition" of points on an elliptic curve provide a much harder analogue of the discrete logarithm problem (which means smaller key sizes)
- Will soon be (somewhat) obsolete due to quantum computing, but some applications may remain in isogeny based cryptography

1.5 Elliptic Curves



Outline

2.1 Zero-Knowledge Proofs

- Beginnings of FHE

2.1 Zero-Knowledge Proofs

- Beginnings of FHE
- Want to construct proof of knowledge of $x \mid f(x) = y$ without revealing x

2.2 Polynomial Commitments

- Want proof that a given (complicated) computation was faithfully carried out by an untrusted agent, without performing the computation itself

2.2 Polynomial Commitments

- Want proof that a given (complicated) computation was faithfully carried out by an untrusted agent, without performing the computation itself
- Specifically in the context of evaluating a (very large) polynomial for an arbitrary input

2.3 Garbled Circuits

- Want to perform multi-party computation on private data

2.3 Garbled Circuits

- Want to perform multi-party computation on private data

Example

Say Alice and Bob want to compare salaries without revealing any information. Need to compute $f(a, b)$ which indicates whether $a > b$, $a < b$, or $a = b$ where a is Alice's salary and b is Bob's.

2.3 Garbled Circuits

- Curry $f(a,b)$ to $f(a)(b)$. Alice computes $f(a)$ and "garbles" the circuit before sending it to Bob, who evaluates $\mathrm{Enc}(f(a)(b))$.

2.3 Garbled Circuits

- Curry $f(a, b)$ to $f(a)(b)$. Alice computes $f(a)$ and “garbles” the circuit before sending it to Bob, who evaluates $\mathrm{Enc}(f(a)(b))$.
- Only Alice can decrypt the result, so Bob cannot peer into the internals of $f(a)$ and therefore never learns a

2.3 Garbled Circuits

- Curry $f(a, b)$ to $f(a)(b)$. Alice computes $f(a)$ and "garbles" the circuit before sending it to Bob, who evaluates $\text{Enc}(f(a)(b))$.
- Only Alice can decrypt the result, so Bob cannot peer into the internals of $f(a)$ and therefore never learns a
- Alice can send a proof that the garbled circuit $\text{Enc}(f(a))$ is an encrypted version f, so Bob can verify that Alice will not gain any information other than the output of f

Outline

3.1 Local Computation

- Full Disk Encryption (LUKS)

3.1 Local Computation

- Full Disk Encryption (LUKS)
- Email Encryption (GPG key signing)

3.1 Local Computation

- Full Disk Encryption (LUKS)
- Email Encryption (GPG key signing)
- Communication Protocols (should be E2EE)

3.2 Vulnerabilities and Paradoxes

- The Ken Thompson Hack (KTH)

3.2 Vulnerabilities and Paradoxes

- The Ken Thompson Hack (KTH)
- Hardware attacks (Thunderspy, evil maid, etc.)

3.2 Vulnerabilities and Paradoxes

- The Ken Thompson Hack (KTH)
- Hardware attacks (Thunderspy, evil maid, etc.)
- Common software vulnerabilities (memory attacks, timing attacks)

4. References

Davidgothberg. 2007. *Merkle-Damgård Hash Construction That Is Used inside Almost All Modern Cryptographic Hash Functions.*. https://commons.wikimedia.org/wiki/File:Merkle-Damgard_hash_big.svg.

Ferran, Lee. 2014. "Ex-NSA Chief: "We Kill People Based on Metadata". American Broadcasting Company. May 2014. http://abcnews.go.com/blogs/headlines/2014/05/ex-nsa-chief-we-kill-people-based-on-metadata.

Helix84, User:Jorge Stolfi based on Image:Hash_function svg by. 2008. *Deutsch: Zeigt Eine Typische Kryptologische Hashfunktion (SHA-1) Am Werk. Beachte, Dass Kleine Unterschiede in Der Eingabe Die Ausgabewerte Drastisch Verändern.*. https://commons.wikimedia.org/wiki/File:Cryptographic_Hash_Function.svg.

4. References

Vinck, A.J. 2011. *English: Illustration of the Idea of the Diffie-Hellman Key Exchange.*. https://commons.wikimedia.org/wiki/File:Diffie-Hellman_Key_Exchange.svg.