Cryptography: Authentication and Future Developments

Alex Corner

Sheffield Hallam University

Wrapping Things Up

- Hashing, Authentication, and Digital Signatures
- Elliptic Curve Cryptography
- Quantum Threats
- What else to explore

Hashing

- A **hash function** is way to take a message as input (usually padded to be fixed size) and produce a fixed-size output, called a **hash** or **message digest**.
- The hash should be a much smaller size than the original message, so that it is fast to compute and can be used as a quick way to authenticate or sign encrypted messages.

Hashing

- A **hash function** is way to take a message as input (usually padded to be fixed size) and produce a fixed-size output, called a **hash** or **message digest**.
- The hash should be a much smaller size than the original message, so that it is fast to compute and can be used as a quick way to authenticate or sign encrypted messages.
- An ideal hash function would have the following properties:
  1. **Pre-Image Resistance:** Given a hash value h, it should be hard to find a message m with $\#(m) = h$.
  2. **Second Pre-Image Resistance:** Given a message m_1 with hash $\#(m_1)$, it should be difficult to find a second message m_2 such that $\#(m_1) = \#(m_2)$.
  3. **Collision Free:** It should be difficult to find two message values $m_1 \neq m_2$ such that $\#(m_1) = \#(m_2)$

Hashing

- A **hash function** is way to take a message as input (usually padded to be fixed size) and produce a fixed-size output, called a **hash** or **message digest**.
- The hash should be a much smaller size than the original message, so that it is fast to compute and can be used as a quick way to authenticate or sign encrypted messages.
- An ideal hash function would have the following properties:
  1. **Pre-Image Resistance:** Given a hash value h, it should be hard to find a message m with $\#(m) = h$.
  2. **Second Pre-Image Resistance:** Given a message m_1 with hash $\#(m_1)$, it should be difficult to find a second message m_2 such that $\#(m_1) = \#(m_2)$.
  3. **Collision Free:** It should be difficult to find two message values $m_1 \neq m_2$ such that $\#(m_1) = \#(m_2)$
- Common cryptographic hashes:
  - Insecure: `MD5`, `SHA-1`,
  - Recommended: `SHA-2`, `SHA-3`.

Hash: Example

Define a hash function on message values $m \in \mathbb{N}$ by:

1. Convert the number to binary and break into 8-bit blocks.
2. Stack up the blocks, padding the first block with 0s if needed.
3. `XOR` the columns to get a new 8-bit number.

(*Don't actually use this as a hash function.*)

E.g., we could have message value $m = 123456$ which is

$$0000\,0111\,0101\,1011\,1100\,1101\,0001\,0101$$

in binary, with padding at the front.

Stacked in blocks:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Authentication Using Public Keys

Setup: Alice publishes a public key, allowing anybody to send her messages encrypted using this public key.

- Problem:
  - Alice wants to receive messages from Bob.
  - She suspects that somebody (Mallory) is intercepting and altering the messages that are sent, or discarding Bob's messages and pretending to be Bob by sending their own.
  - How does Alice authenticate Bob's messages?
- Solution:
  - Bob sends his encrypted messages along with a hash of the original message.
  - Hashing ensures that it is difficult for Mallory to find a new message that has the same hash as Bob's message.

Signing Using Public Keys

Setup: Alice publishes a public key, allowing anybody to send her messages encrypted using this public key.

- New Problem:
  - Mallory discards Bob's messages *and* hash, and replaces it with their own message and hash.
- Solution:
  - Bob encrypts his message to Alice using Alice's public key.
  - But now Bob sets up his own public key.
  - He **signs** the message by using his private key to 'decrypt' the hash.
  - Alice can decrypt the message (using her private key) and 'encrypt' the hash to check it (using Bob's public key).
  - Mallory can still intercept the message and hash, but can't create a 'signature' to put in place of Bob's.
- Alternative Use: The original message might not need to be encrypted, but it might still need to be 'signed' by Bob. The same method can be used here.

Authentication and Encryption

$$\begin{tikzcd}
(N_B, e_B) & (N_A, e_A) \\
\text{Bob} \arrow[r] & \text{Alice} \\
m \arrow[r, mapsto] & E_A(m, e_A) \\
m \arrow[r, mapsto] & (E_A(m, e_A), \#(m)) \\
m \arrow[r, mapsto] & (E_A(m, e_A), D_B(\#(m), e_B))
\end{tikzcd}$$

Example: Digital Signatures with RSA

- Alice has public RSA key $(n_A, e_A) = (187, 7)$ (private: $d_A = 23$) and Bob has public RSA key $(n_B, e_B) = (299, 17)$ (private: $d_B = 233$).
- They agree to use the number of binary bits as a hashing function (*don't use this to hash things either*). E.g.,

$$\#(123) = \#(0111\,1011_2) = 6.$$

- Bob wants to send the message $m = 25$ to Alice. The hash is $\#(25) = \#(0001\,1001_2) = 3$.
- Bob encrypts m using Alice's key: $m^{e_A} = 25^7 \mod 187 = 185$.
- Bob decrypts $\#(m)$ using his private key: $\#(m)^{d_B} = 3^{233} \mod 299 = 9$.
- Bob sends $(185, 9)$ to Alice.
- Alice decrypts the message using her private key: $185^{d_A} = 185^{23} \mod 187 = 25$.
- Alice encrypts the hash using Bob's public key: $9^{e_B} = 9^{17} \mod 299 = 3$.
- Alice checks that $\#(185) = 3$ and is happy that the message is from Bob.

PGP: Pretty Good Privacy

- These sorts of ideas are in constant use.
- Pretty Good Privacy was developed by Phil Zimmerman in the late 1980s/early 1990s.
- It is primarily used to authenticate and encrypt emails.
- The principle is that a **web of trust** is built up by the trust users give each other.
- https://pgp.mit.edu/
- http://www.faqs.org/faqs/pgp-faq/part1/

Elliptic Curve Cryptography

- RSA is incredibly secure, but also incredibly inefficient. Can we achieve a similar level of security, but using much smaller keys?

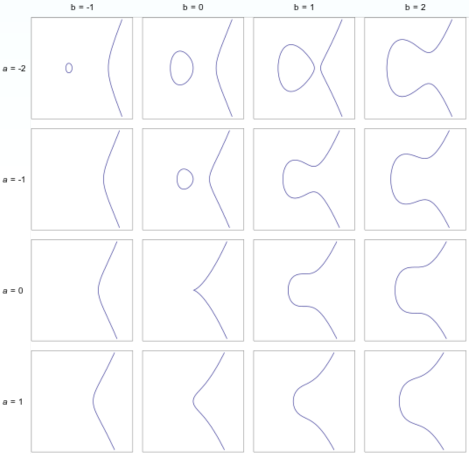
Elliptic Curve Cryptography

- RSA is incredibly secure, but also incredibly inefficient. Can we achieve a similar level of security, but using much smaller keys?
- In 1985, another public key system was developed by Victor S. Miller and Neal Koblitz, known as elliptic curve cryptography.

Elliptic Curve Cryptography

- RSA is incredibly secure, but also incredibly inefficient. Can we achieve a similar level of security, but using much smaller keys?
- In 1985, another public key system was developed by Victor S. Miller and Neal Koblitz, known as elliptic curve cryptography.
- It purports to give similar levels of security as RSA-4096, but only using a 313-bit key.

Elliptic Curve Cryptography

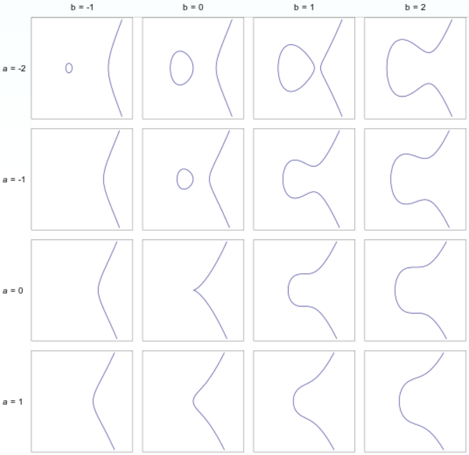


- An **elliptic curve** is the set of solutions to an equation of the form

$$y^2 = x^3 + ax + b,$$

along with a point at infinity: ∞.

Elliptic Curve Cryptography

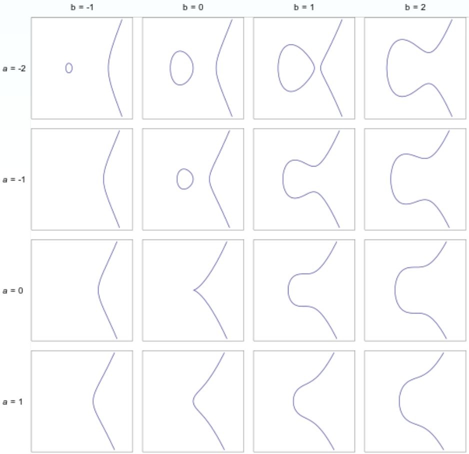


- An **elliptic curve** is the set of solutions to an equation of the form

  $$y^2 = x^3 + ax + b,$$

  along with a point at infinity: ∞.
- These curves have turned up in various parts of mathematics since at least the third century, in the study of Diophantine equations: equations where we are interested in whole number or fractional solutions to polynomials.

Elliptic Curve Cryptography



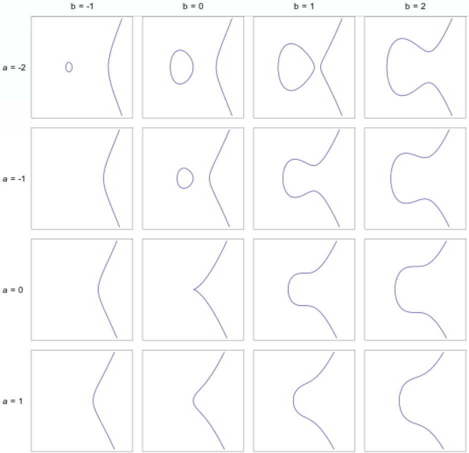
- An **elliptic curve** is the set of solutions to an equation of the form

$$y^2 = x^3 + ax + b,$$

along with a point at infinity: ∞.

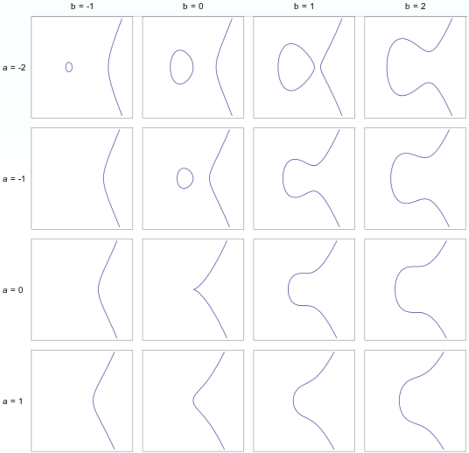
- These curves have turned up in various parts of mathematics since at least the third century, in the study of Diophantine equations: equations where we are interested in whole number or fractional solutions to polynomials.
- They also were a major component in Andrew Wile's proof of Fermat's Last Theorem.

Elliptic Curve Cryptography



- As you can see, these curves do not look like ellipses. They turn up in the study of *elliptic integrals*, which are methods to find formulas for arc lengths of ellipses.

Elliptic Curve Cryptography

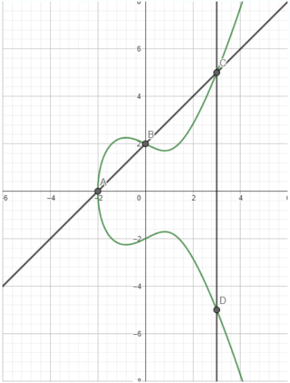


- As you can see, these curves do not look like ellipses. They turn up in the study of *elliptic integrals*, which are methods to find formulas for arc lengths of ellipses.
- Taking two points on an elliptic curve, we can add them together. But this isn't just

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2).$$

We add them together in a slightly strange geometric fashion. This is what will lead us to using elliptic curves for encryption.

Adding Points on Elliptic Curves



Adding $A = (-2, 0)$ to $B = (0, 2)$ results in $D = (3, -5)$.

Elliptic Curves in Practice

Elliptic curves took a bit longer to take off as a trusted encryption scheme, it often being said that the mathematics put people off as opposed to RSA's fairly straightforward algorithms. But the takeup of Elliptic Curve Cryptography has been growing in the last couple of decades. A publically available elliptic curve is 'Curve25519'.

- https://www.wikipedia.org/ now uses ECC;

Elliptic Curves in Practice

Elliptic curves took a bit longer to take off as a trusted encryption scheme, it often being said that the mathematics put people off as opposed to RSA's fairly straightforward algorithms. But the takeup of Elliptic Curve Cryptography has been growing in the last couple of decades. A publically available elliptic curve is 'Curve25519'.

- https://www.wikipedia.org/ now uses ECC;
- Bitcoin addresses are derived from elliptic curve public keys, with transactions being authorised using digital signatures;

Elliptic Curves in Practice

Elliptic curves took a bit longer to take off as a trusted encryption scheme, it often being said that the mathematics put people off as opposed to RSA's fairly straightforward algorithms. But the takeup of Elliptic Curve Cryptography has been growing in the last couple of decades. A publically available elliptic curve is 'Curve25519'.

- https://www.wikipedia.org/ now uses ECC;
- Bitcoin addresses are derived from elliptic curve public keys, with transactions being authorised using digital signatures;
- Many communications on social media are encrypted using the Signal Protocol: a combination of ECC (Curve25519), AES-256, and HMAC-SHA256;

Quantum Attacks

- The advent of quantum computation may open up many more avenues for cryptanalysis. One algorithm in particular is a bit of a shady figure for cryptographers: Shor's Algorithm.

Quantum Attacks

- The advent of quantum computation may open up many more avenues for cryptanalysis. One algorithm in particular is a bit of a shady figure for cryptographers: Shor's Algorithm.
- Shor's Algorithm is an algorithm developed by Peter Shor in the 1990s, which would allow efficient factorisation of numbers into their constituent primes. He also developed other algorithms, one of which tackles the discrete logarithm problem (the security behind Diffie-Hellman key exchange).

Quantum Attacks

- The advent of quantum computation may open up many more avenues for cryptanalysis. One algorithm in particular is a bit of a shady figure for cryptographers: Shor's Algorithm.
- Shor's Algorithm is an algorithm developed by Peter Shor in the 1990s, which would allow efficient factorisation of numbers into their constituent primes. He also developed other algorithms, one of which tackles the discrete logarithm problem (the security behind Diffie-Hellman key exchange).
- Grover's Algorithm, mentioned in the AES section, is another quantum algorithm, but essentially makes the problem of *function inversion* much more efficient. This could potentially allow keys for schemes such as AES to be broken much more easily, massively speeding up brute force attacks.

Quantum Attacks

- The advent of quantum computation may open up many more avenues for cryptanalysis. One algorithm in particular is a bit of a shady figure for cryptographers: Shor's Algorithm.
- Shor's Algorithm is an algorithm developed by Peter Shor in the 1990s, which would allow efficient factorisation of numbers into their constituent primes. He also developed other algorithms, one of which tackles the discrete logarithm problem (the security behind Diffie-Hellman key exchange).
- Grover's Algorithm, mentioned in the AES section, is another quantum algorithm, but essentially makes the problem of *function inversion* much more efficient. This could potentially allow keys for schemes such as AES to be broken much more easily, massively speeding up brute force attacks.
- Were a practical quantum computer to be developed, then these algorithms would pose massive threats to the security of RSA, AES, and ECC.

Topics to Explore

We have only covered a tiny fraction of cryptography and related topics. Here are some things to look into if you want to explore this even further:

- Secret-sharing schemes
- Cryptocurrency
- Types of attack:
  - timing, meet-in-the-middle, man-in-the-middle, replay, side-channel,
  - known plaintext, chosen plaintext, known ciphertext, common modulus,
  - differential cryptanalysis, boomerang
- Security in electronic voting schemes
- Quantum and post-quantum cryptography
- The history of codebreaking: check the reading list
- The history of Alice and Bob
- Tanja Lange's video playlists on Cryptology

Tutorials

In the tutorials this week:

- Lots of questions about hashing and authentication, using RSA and other algorithms. This is good revision practice for understanding RSA.
- Some more revision practice for using RSA.