NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Seminar

Cryptographic Constructions

Sourav SEN GUPTA

Lecturer, SCSE, NTU

Deep-Dive

Hash Functions

Cryptographic Hash Functions

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

- Efficiently computable : Expected complexity $O(m)$ for an m-bit input
- Preimage resistance : Finding input x given $y = H(x)$ is not possible
- 2nd Preimage resistance : Given x, finding $y \neq x$ with $H(y) = H(x)$ is impossible
- Collision resistance : Finding any pair $x \neq y$ with $H(x) = H(y)$ is impossible

Really?! Can anything be "impossible" in a finite computation context?

Preimage Resistance

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

Finding input x given $y = H(x)$ is computationally infeasible
Analogy : Given some birthday, would I be able to determine whose it is?

An attack should be no better than "randomly sampling" the input space.

Probability of matching the given output should thus be 2^{-n} in this case.

2^{nd} Preimage Resistance

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

Given x, finding $y \neq x$ with $H(y) = H(x)$ is computationally infeasible
Analogy : Given your birthday, find another person with the same birthday.

An attack should be no better than "randomly sampling" the input space.
Probability of matching hash of given input should thus be 2^{-n} in this case.

Collision Resistance

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

Finding any pair $x \neq y$ with $H(x) = H(y)$ is computationally infeasible

Analogy : Finding any two persons with the exact same birthday.

An attack should be no better than "randomly sampling" the input space.

Birthday Paradox : Probability of finding colliding input pairs is quite high!

Birthday Paradox

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

Analogy : Finding any two persons with the exact same birthday.

What is the probability that there is <u>no collision</u> after checking with m persons?

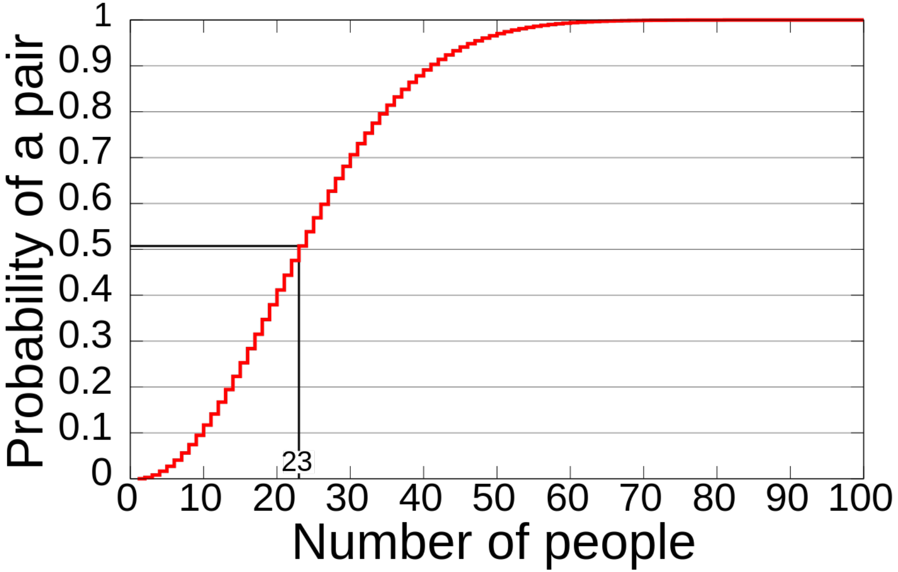
Birthday Paradox

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$H : \{0,1\}^* \rightarrow \{0,1\}^n$

Analogy : Finding any two persons with the exact same birthday.

What is the probability that there is a collision after checking with *m* persons randomly?



Collision Resistance

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

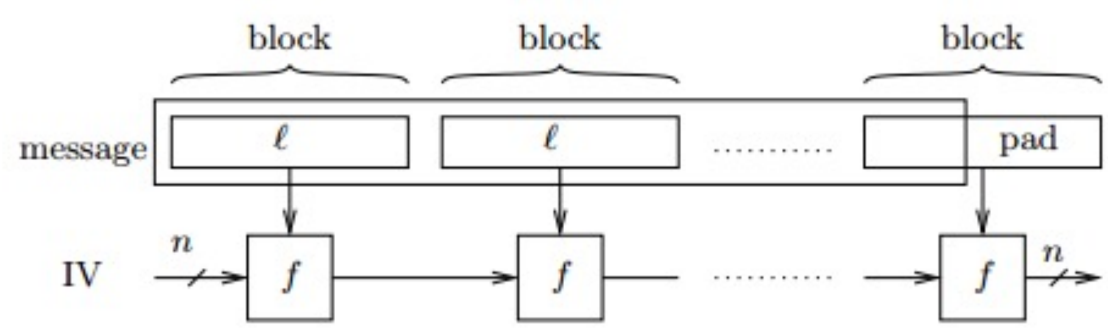
$H : \{0,1\}^* \rightarrow \{0,1\}^n$

Finding any pair $x \neq y$ with $H(x) = H(y)$ is computationally infeasible
Analogy : Finding any two persons with the exact same birthday.

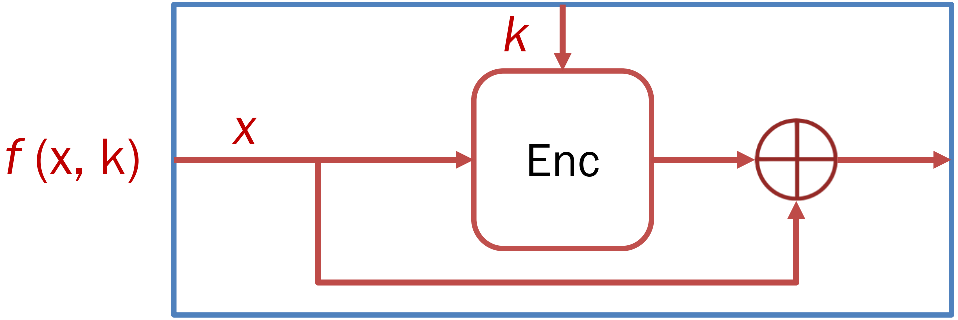
An attack should be no better than "birthday case" in the input space.

Probability of collision in hash should thus be bounded by $2^{-(n/2)}$ (approx).

SHA-256 Hash Function

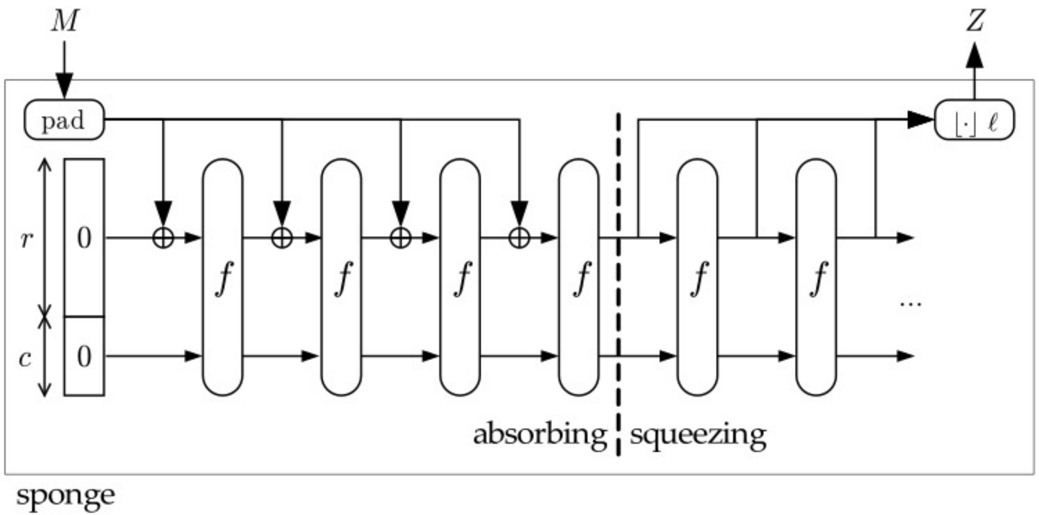


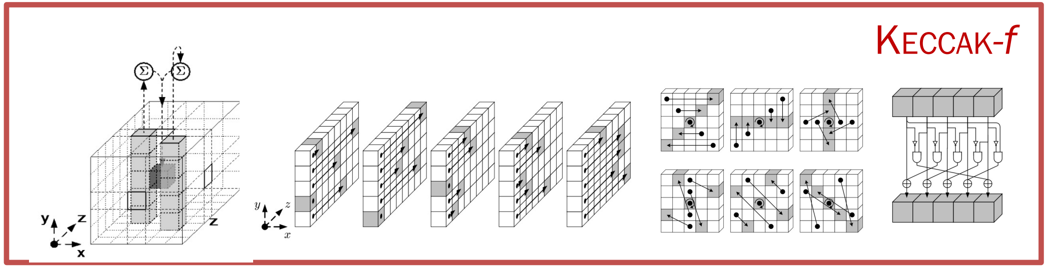
Merkle–Damgård Construction
Hash from Compression Function



Davies-Meyer Construction
Compression from Block Cipher

keccak256 Hash Function





Sponge Construction
Hash from Permutations

Keccak-*f* Permutations
Operations on a Finite State

Puzzle Friendliness

Function from arbitrary Domain to fixed Range
(arbitrary sized message to fixed length digest)

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

Finding *x*, part of the input, given $y = H(k \mid\mid x)$ is computationally infeasible
if *k*, the other part of the input is chosen from a high min-entropy distribution.

An attack should be no better than "randomly sampling" the input space.

Probability of matching the given output should thus be 2^{-n} in this case.

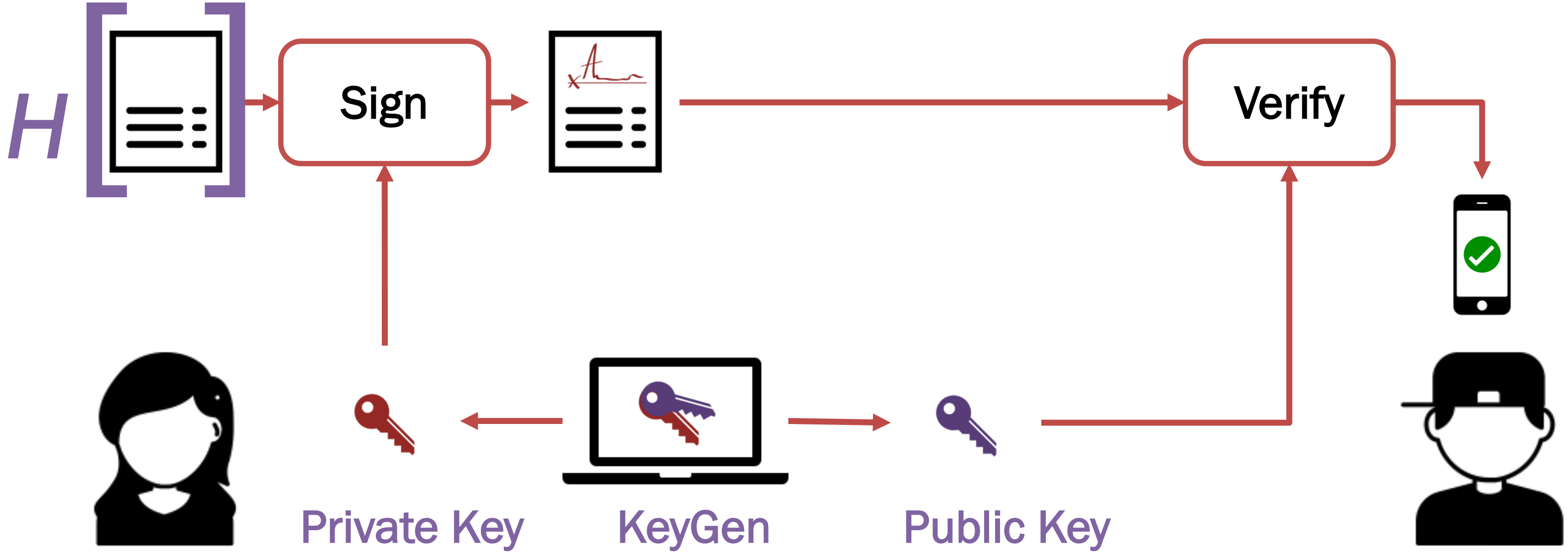
This will be used for **Reusable Proof-of-Work** and other **Search Puzzles.**

Deep-Dive

Digital Signature

Digital Signature

Comprises of two stages : **Sign and Verify**



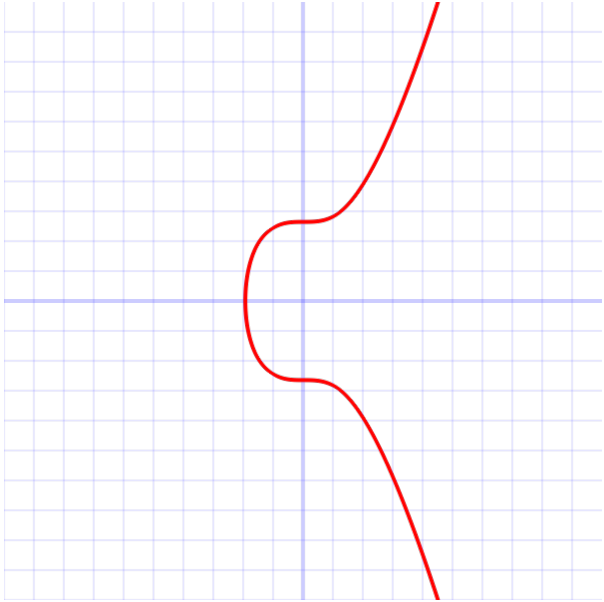
ECDSA Digital Signature

Elliptic Curve is the underlying Algebraic Structure

Bitcoin uses a specific set of ECDSA parameters

- Standard NIST elliptic curve **secp256k1**
- 256-bit Private Key, 512-bit Public Key
- 256-bit Message, 512-bit Signature

SHA256 perfectly suits the input specification.



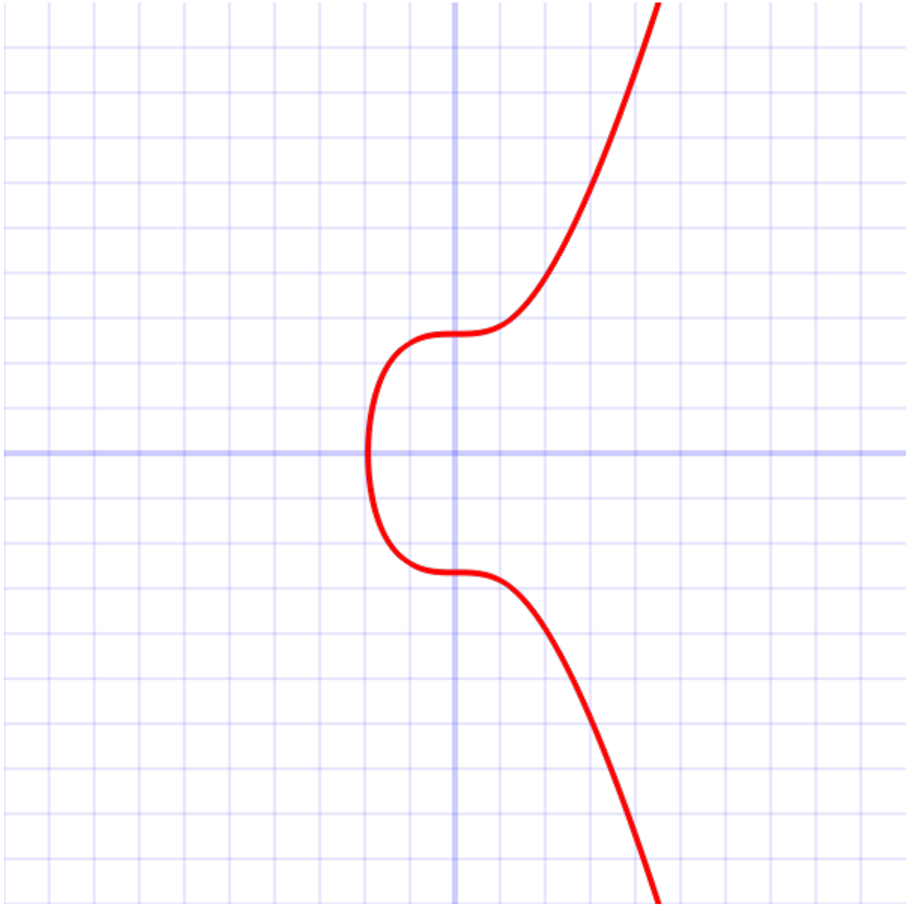
Curve secp256k1 over Reals : $y^2 = x^3 + 7$

Elliptic Curve

secp256k1 : $y^2 = x^3 + 7$ (not yet)

- Cubic curve over the Real Field
- Continuous over the Real Field
- Contains infinite Set of Points

- "Addition" is defined on Points
- Hence, "Scalar Multiplication"



Online tool : https://andrea.corbellini.name/ecc/interactive/reals-add.html

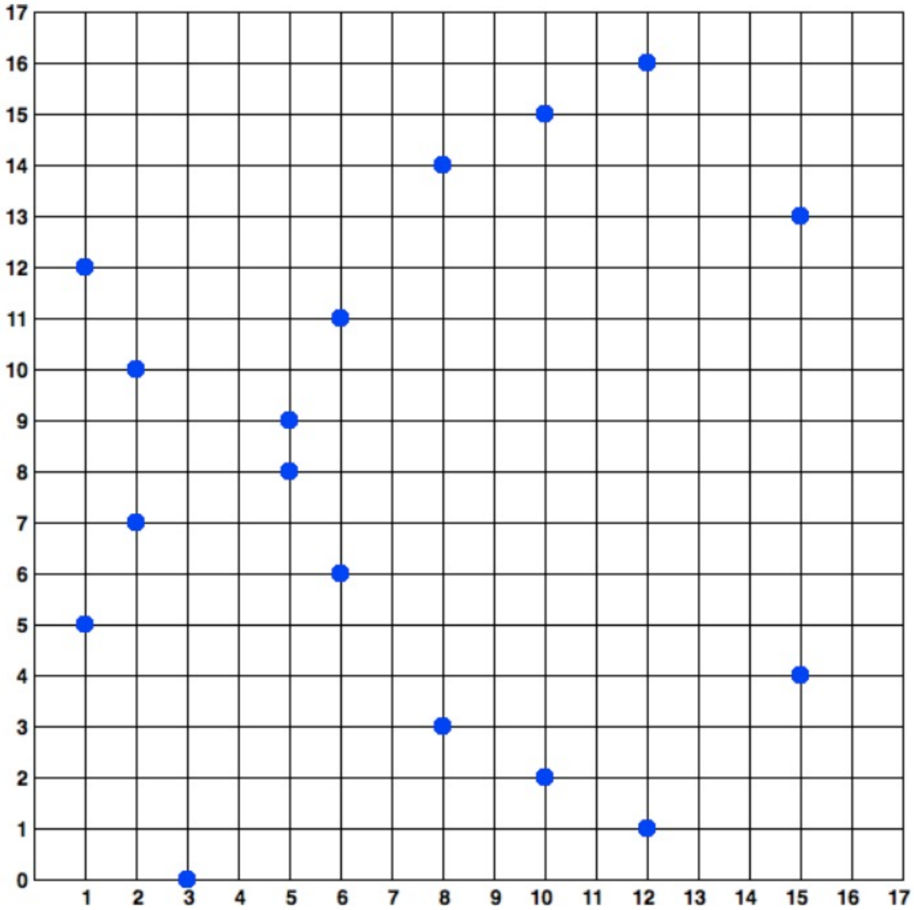
Elliptic Curve

secp256k1 : $y^2 = x^3 + 7$ over F_p

F_p : Prime Field (mod p = 17) (almost)

- Discrete over the Finite Fields
- Contains finite Group of Points

- "Addition" follows from Reals
- Hence, "Scalar Multiplication"



Online tool : https://andrea.corbellini.name/ecc/interactive/modk-add.html

Elliptic Curve

secp256k1 : $y^2 = x^3 + 7$ over F_p

F_p : Prime Field (256-bit prime p)

- Weierstrass normal form (simple)
- Contains a finite "Group of Points"
- Including Point at Infinity (identity)

- Addition and Scalar Multiplication
- "Hard" Discrete Logarithm Problem

Koblitz curve in Weierstrass normal form T(*p, a, b, G, n, h*)

p	=		FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
			FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
	=		$2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
E	=		$\{(x,y) : y^2 = x^3 + 0x + 7 \bmod p\}$ a = 0, b = 7
G	=	02	79BE667E F9DCBBAC 55A06295 CE870B07
			029BFCDB 2DCE28D9 59F2815B 16F81798
n	=		FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE
			BAAEDCE6 AF48A03B BFD25E8C D0364141
h	=		01

Curve parameters : http://www.secg.org/sec2-v2.pdf

Discrete Logarithm Problem

DLP : Given *a* and a^x, find *x*

DLP is not so hard in most groups

Example : Given *a* and *x*a*, find *x*

ECDLP : Given *G* and *k*G*, find *k*

Computationally “hard” for secp256k1

We can set **PriKey = *k*** (random, secret)

and corresponding **PubKey = *K* = *k*G***

Koblitz curve in Weierstrass normal form T(*p, a, b, G, n, h*)

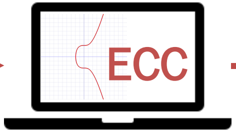
p	=	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
	=	$2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
E	=	$\{(x,y) : y^2 = x^3 + 0x + 7 \bmod p\}$ a = 0, b = 7
G	=	02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
n	=	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
h	=	01

DLP : https://en.wikipedia.org/wiki/Discrete_logarithm

Bitcoin Signature (ECDSA)

Key Generation : Elliptic Curve Cryptography

secp256k1 : Koblitz curve defined by T(*p, a, b, G, n, h*)

k → 

→ $K = k*G$

Private Key	KeyGen	Public Key
256-bit integer	secp256k1	512-bit point (or compressed)

Transaction outputs (UTXO) locked using script to K

Transaction outputs (UTXO) unlocked by sign with k

UTXO : Unspent Transaction Output

Amount : Bitcoin in Satoshi

Bitcoin Locking Script
Locks the UTXO to a Public Key
Only the Private Key can unlock

k  K

ECDSA : https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm