Blockchain Security: From Curves to Contracts

Dr. Jimmy Chen, IKV & NTU

Alex Liu, AMIS & MaiCoin

HITCON Pacific 2016

Aspects of Security

- ECDSA for Transaction Signing (including hardware signing)
- Hash Function Collision Resistance
- Privacy Preserving Features (Zero-Knowledge Proofs)
- Consensus Algorithms
- Smart Contract Correctness

Introduction to Blockchain

「區塊鏈」技術進軍華爾街，十年內 200 萬個銀行工作將蒸發

作者 黃 嬿 | 發布日期 2016 年 04 月 10 日 9:00 | 分類 人力資源 , 自動化 G+1 讚 分享 1,298

摩根大通與花旗等銀行成功將比特幣背後的區塊鏈技術 (blockchain) 應用在信用違約交換 (credit default swap, CDS) 市場上，此進展將能讓區塊鏈技術在主流金融領域站穩腳步，幫銀行省下人事成本，花旗報告甚至稱銀行引進自動化未來十年會少 200 萬個銀行工作。

Source: http://technews.tw/2016/04/10/blockchain-applied-on-wall-street

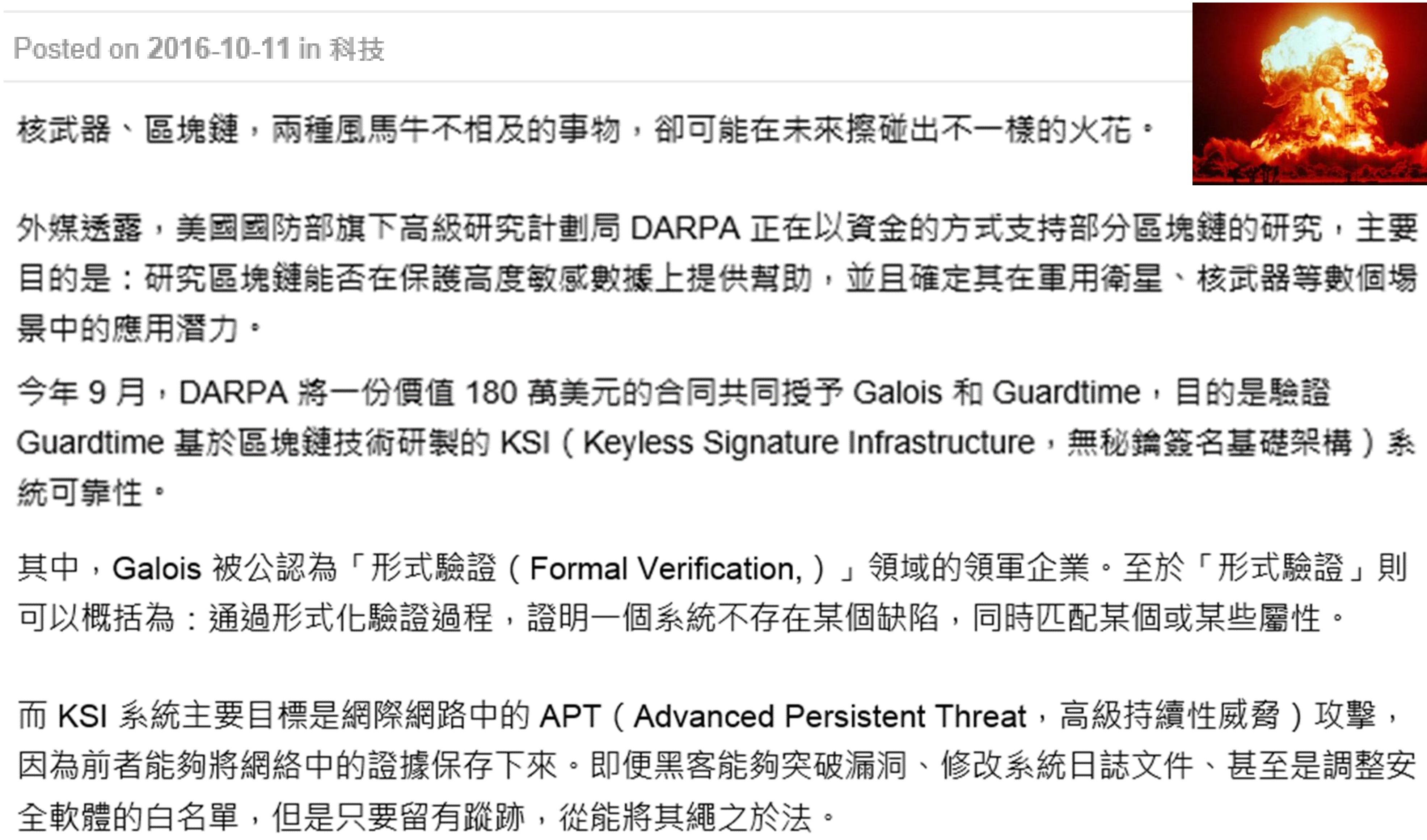
區塊鏈將讓銀行 10 年內消失？比特幣大老：言之過早

作者 MoneyDJ | 發布日期 2016 年 04 月 12 日 16:40 | 分類 支付方案 , 財經 G+1 讚 分享 133

金融科技（Fintech）來勢洶洶，「區塊鏈」（Blockchain）技術更成為顯學，雖然許多業界大老警告銀行的生存恐受威脅，俄羅斯聯邦儲蓄銀行（Sberbank）副總裁 Andrey Sharov 上週更警告銀行可能 2026 年就會因為區塊鏈而全面消失，但比特幣非營利組織「比特幣基金會」（Bitcoin Foundation）董事長 Brock Pierce 卻認為，討論這些都還言之過早。

Source: http://finance.technews.tw/2016/04/12/blockchain-bank-fintech

未來，核武器可能也用區塊鏈進行控制？



Posted on 2016-10-11 in 科技

核武器、區塊鏈，兩種風馬牛不相及的事物，卻可能在未來擦碰出不一樣的火花。

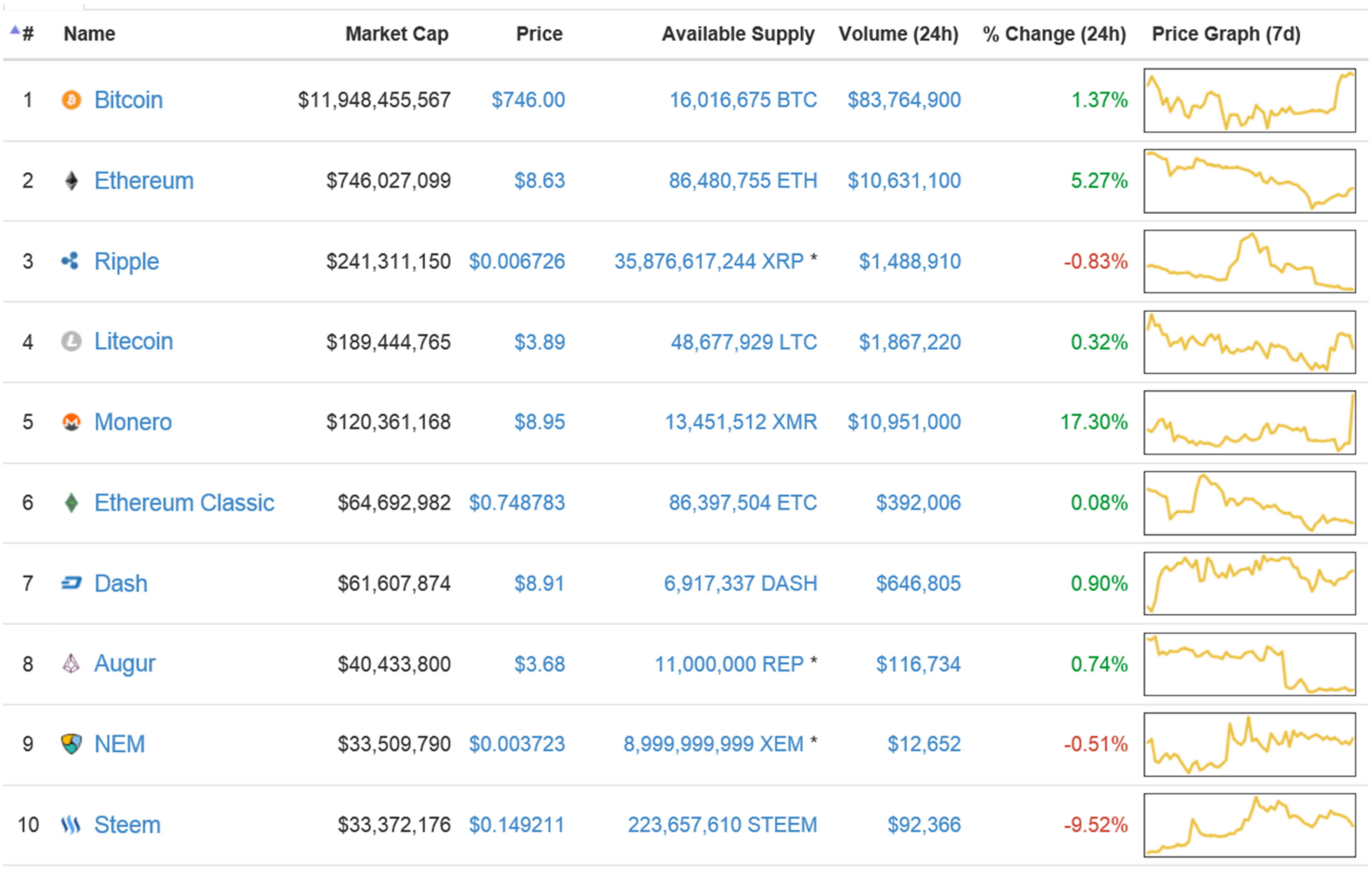
外媒透露，美國國防部旗下高級研究計劃局 DARPA 正在以資金的方式支持部分區塊鏈的研究，主要目的是：研究區塊鏈能否在保護高度敏感數據上提供幫助，並且確定其在軍用衛星、核武器等數個場景中的應用潛力。

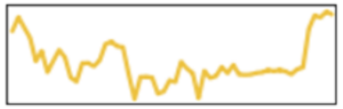
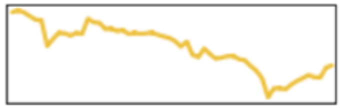
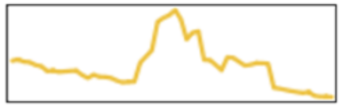
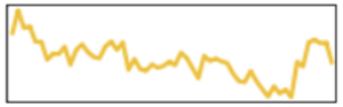
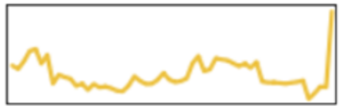
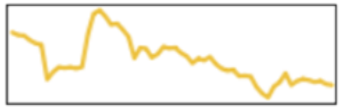
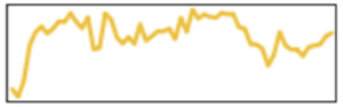
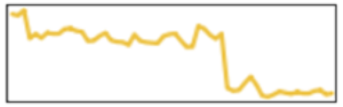
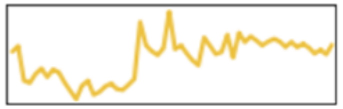
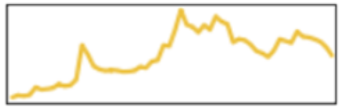
今年 9 月，DARPA 將一份價值 180 萬美元的合同共同授予 Galois 和 Guardtime，目的是驗證 Guardtime 基於區塊鏈技術研製的 KSI（Keyless Signature Infrastructure，無秘鑰簽名基礎架構）系統可靠性。

其中，Galois 被公認為「形式驗證（Formal Verification,）」領域的領軍企業。至於「形式驗證」則可以概括為：通過形式化驗證過程，證明一個系統不存在某個缺陷，同時匹配某個或某些屬性。

而 KSI 系統主要目標是網際網路中的 APT（Advanced Persistent Threat，高級持續性威脅）攻擊，因為前者能夠將網絡中的證據保存下來。即便黑客能夠突破漏洞、修改系統日誌文件、甚至是調整安全軟體的白名單，但是只要留有蹤跡，從能將其繩之於法。

Source: https://kknews.cc/tech/m4kmbp.html

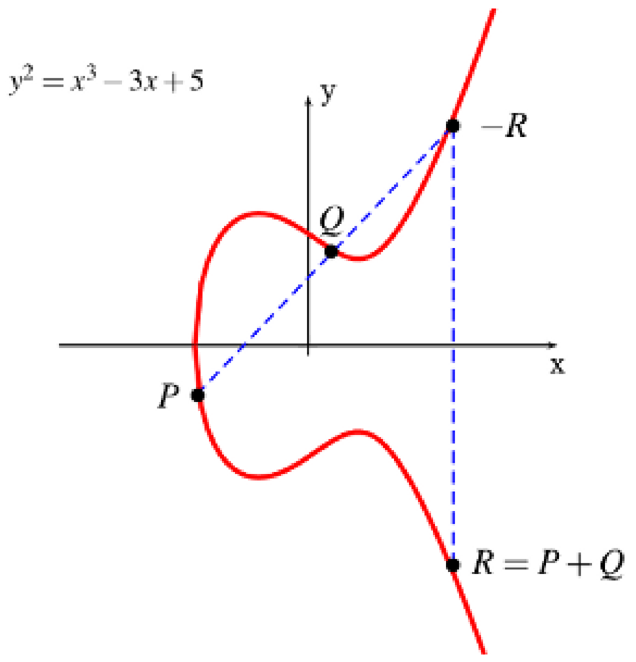


▲#	Name	Market Cap	Price	Available Supply	Volume (24h)	% Change (24h)	Price Graph (7d)
1	Bitcoin	$11,948,455,567	$746.00	16,016,675 BTC	$83,764,900	1.37%	
2	Ethereum	$746,027,099	$8.63	86,480,755 ETH	$10,631,100	5.27%	
3	Ripple	$241,311,150	$0.006726	35,876,617,244 XRP *	$1,488,910	-0.83%	
4	Litecoin	$189,444,765	$3.89	48,677,929 LTC	$1,867,220	0.32%	
5	Monero	$120,361,168	$8.95	13,451,512 XMR	$10,951,000	17.30%	
6	Ethereum Classic	$64,692,982	$0.748783	86,397,504 ETC	$392,006	0.08%	
7	Dash	$61,607,874	$8.91	6,917,337 DASH	$646,805	0.90%	
8	Augur	$40,433,800	$3.68	11,000,000 REP *	$116,734	0.74%	
9	NEM	$33,509,790	$0.003723	8,999,999,999 XEM *	$12,652	-0.51%	
10	Steem	$33,372,176	$0.149211	223,657,610 STEEM	$92,366	-9.52%	

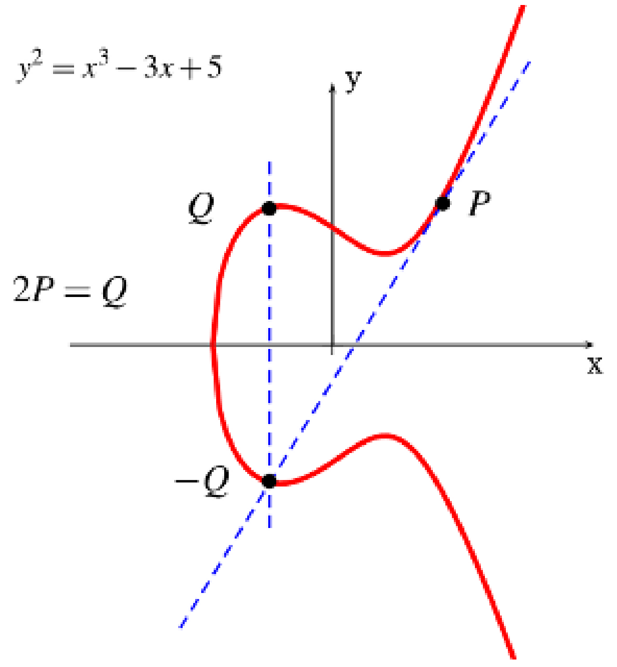
http://coinmarketcap.com

Elliptic Curve 橢圓曲線

- The rich and deep theory of Elliptic Curves has been studied by mathematicians over 150 years
- Elliptic Curve over ***R*** : $y^2 = x^3 + ax + b$



Point Addition



Point Doubling

Image Courtesy: http://www.embedded.com/design/safety-and-security/4396040/An-Introduction-to-Elliptic-Curve-Cryptography

Curves over Prime Fields 質數體上的曲線

Addition:

$(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$

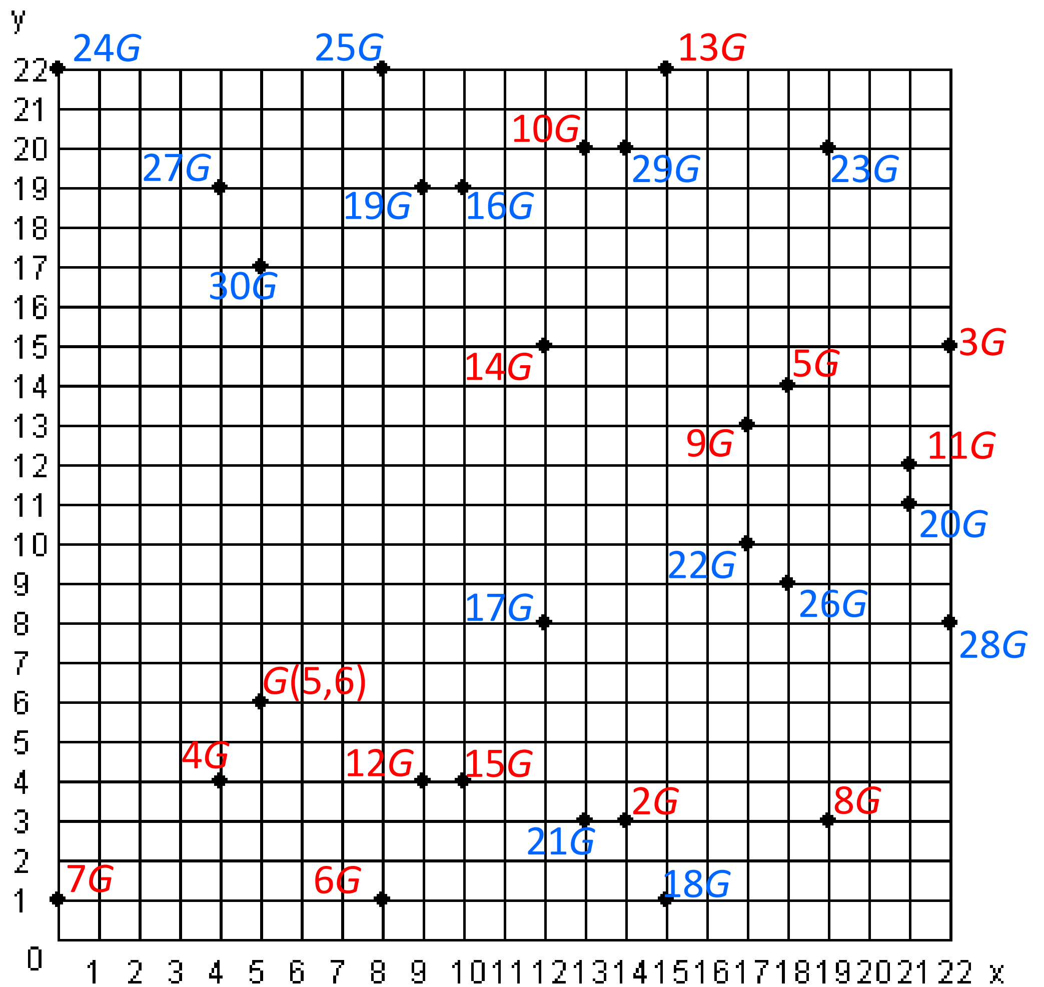
Doubling:

$(x_3, y_3) = [2]\,(x_1, y_1)$

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \bmod p & \text{(addition)} \\ \dfrac{3x_1^2 + a}{2y_1} \bmod p & \text{(doubling)} \end{cases}$$

$$x_3 = s^2 - x_1 - x_2 \mod p$$

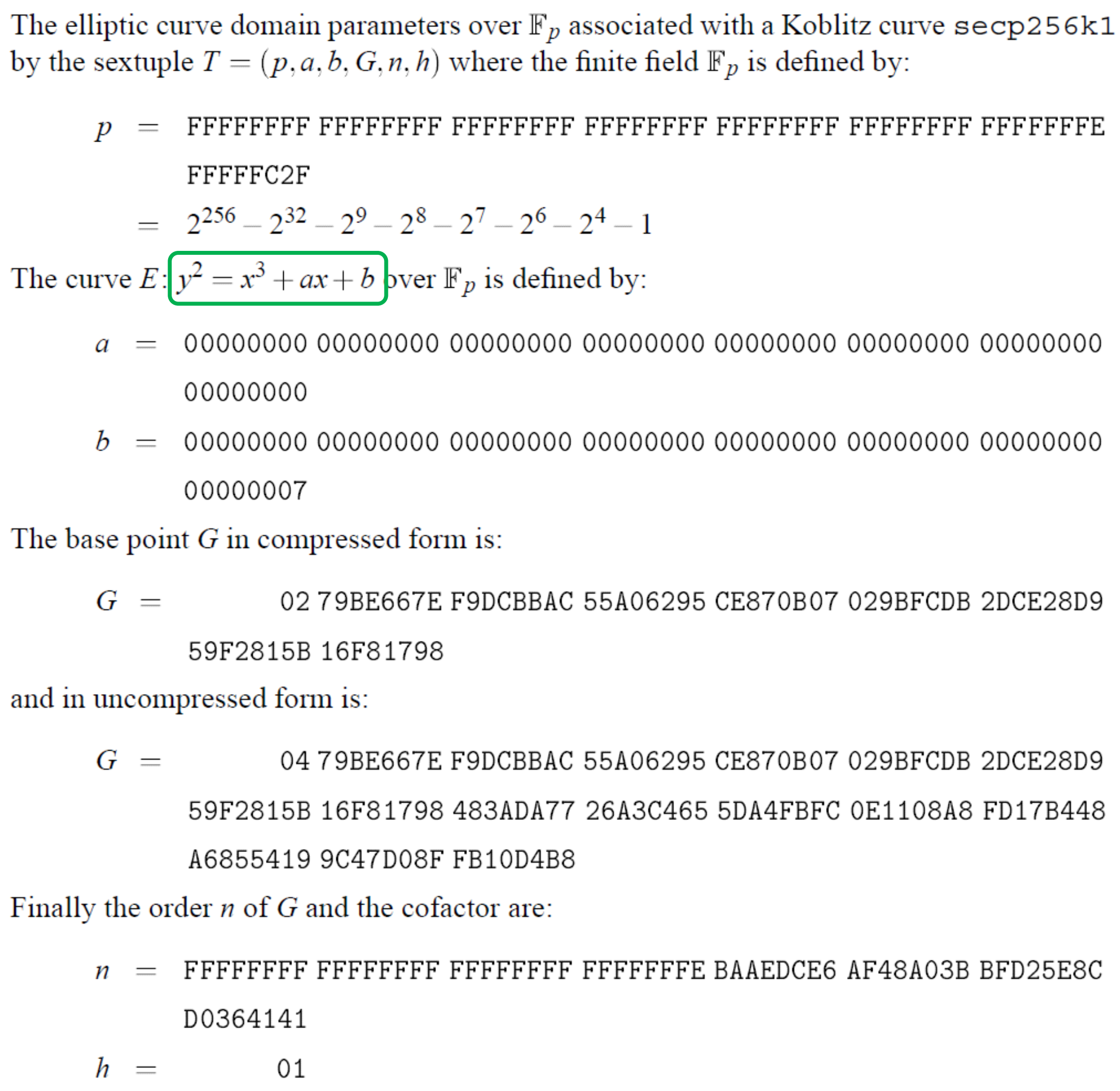
$$y_3 = s(x_1 - x_3) - y_1 \mod p$$



$y^2 = x^3 + 5x + 1$ over F_{23}

30 solutions

The Curve used by Bitcoin and Ethereum



The elliptic curve domain parameters over $\mathbb{F}_p$ associated with a Koblitz curve `secp256k1` are specified by the sextuple $T = (p, a, b, G, n, h)$ where the finite field $\mathbb{F}_p$ is defined by:

p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$

The curve E: $\boxed{y^2 = x^3 + ax + b}$ over $\mathbb{F}_p$ is defined by:

a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007

The base point G in compressed form is:

G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798

and in uncompressed form is:

G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

Finally the order n of G and the cofactor are:

n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

h = 01

256-bit prime

256-bit prime

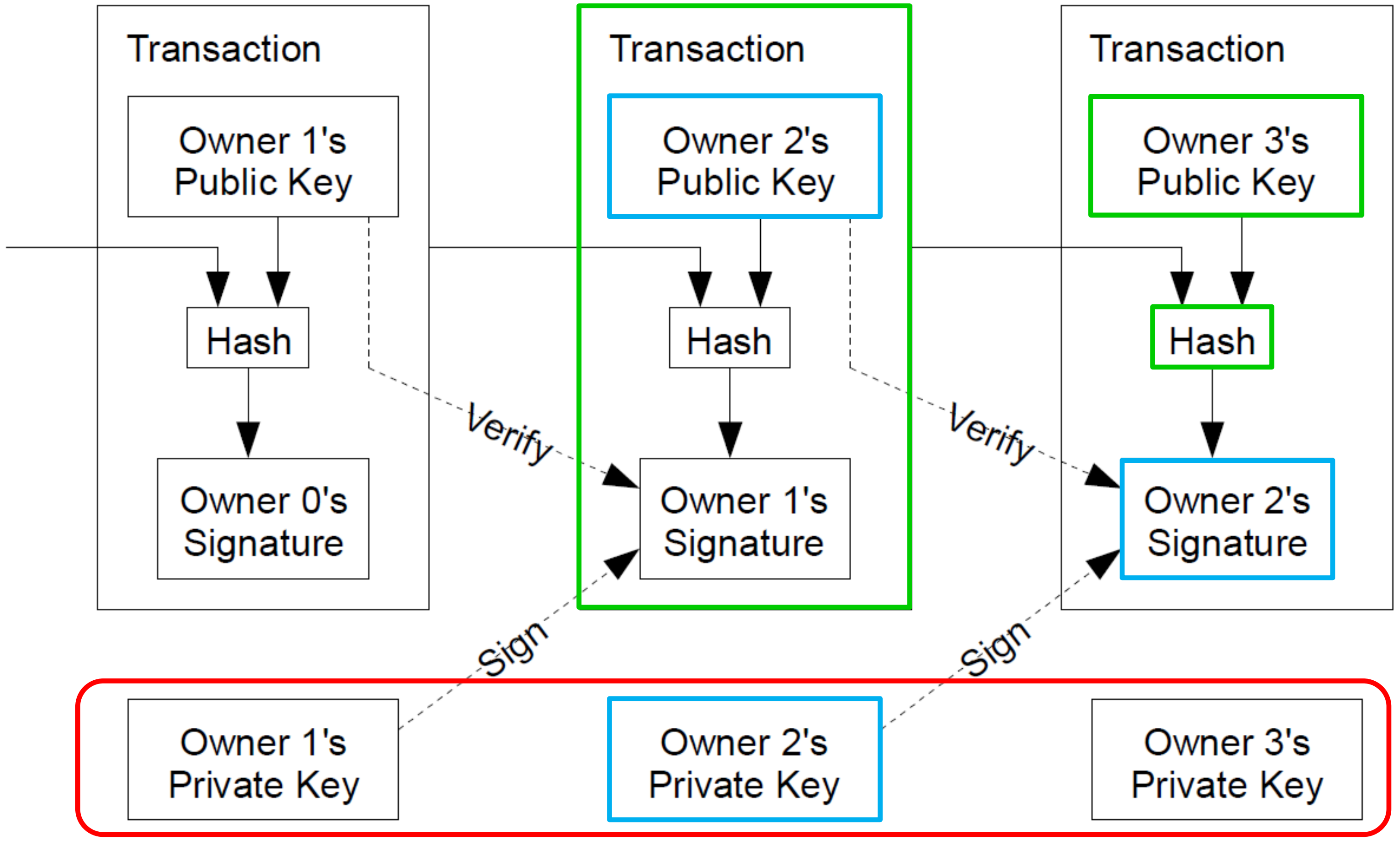
橢圓曲線 secp256k1

https://en.bitcoin.it/wiki/Secp256k1

Key Pairs 金鑰對

- The base point G is fixed on the given Elliptic Curve
- $P = [m]\,G$
  - Given m, it is **easy and fast** to find the point P
    - Using “double and add” for scalar multiplication
  - Given P, it is **extremely hard** to find the integer m
    - Elliptic Curve Discrete Logarithm Problem (橢圓曲線離散對數問題)
  - A randomly generated integer m is a **private key**
    - A private key is used to sign Bitcoin transactions with ECDSA
  - The point P is the **public key** corresponding to m
    - A public key is used by other nodes to verify Bitcoin transactions
    - **A Bitcoin address is the hash value of a public key *P***

Bitcoin Transactions 交易

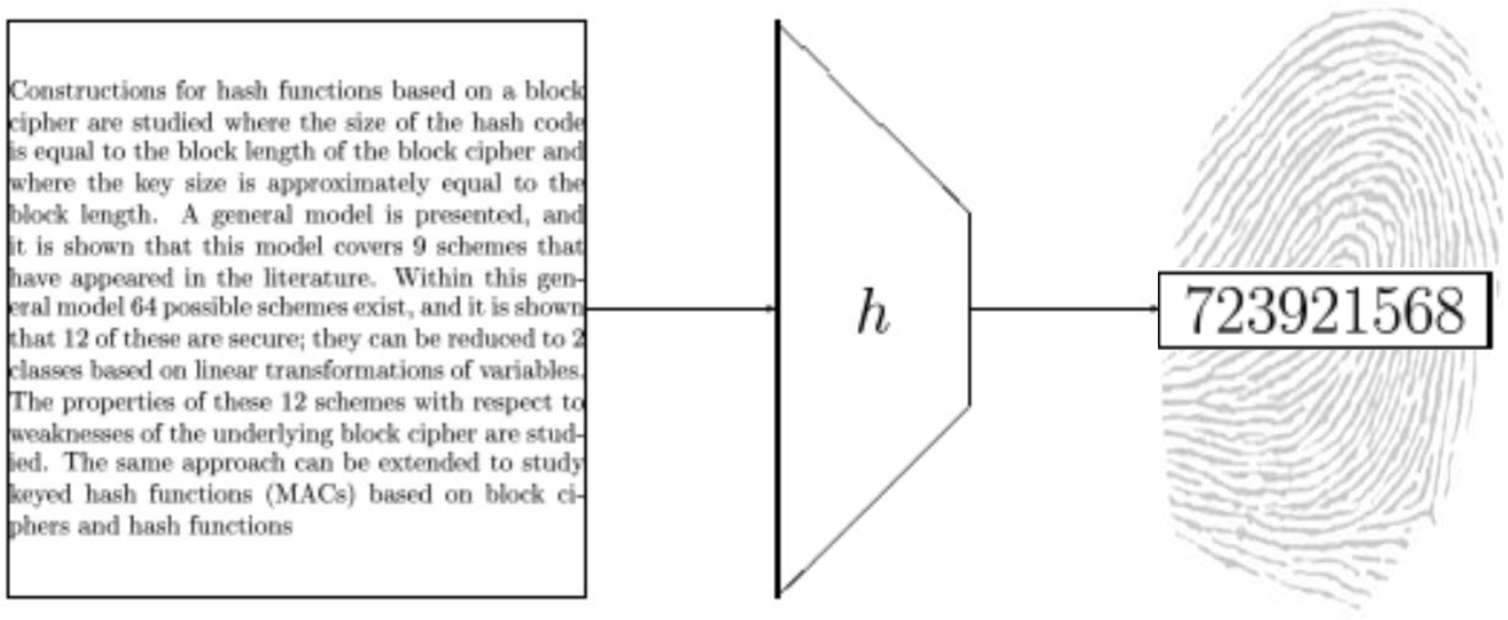


Must be protected very well!!!

http://bitcoin.org/bitcoin.pdf 中本聰

Hash Functions 雜湊函數

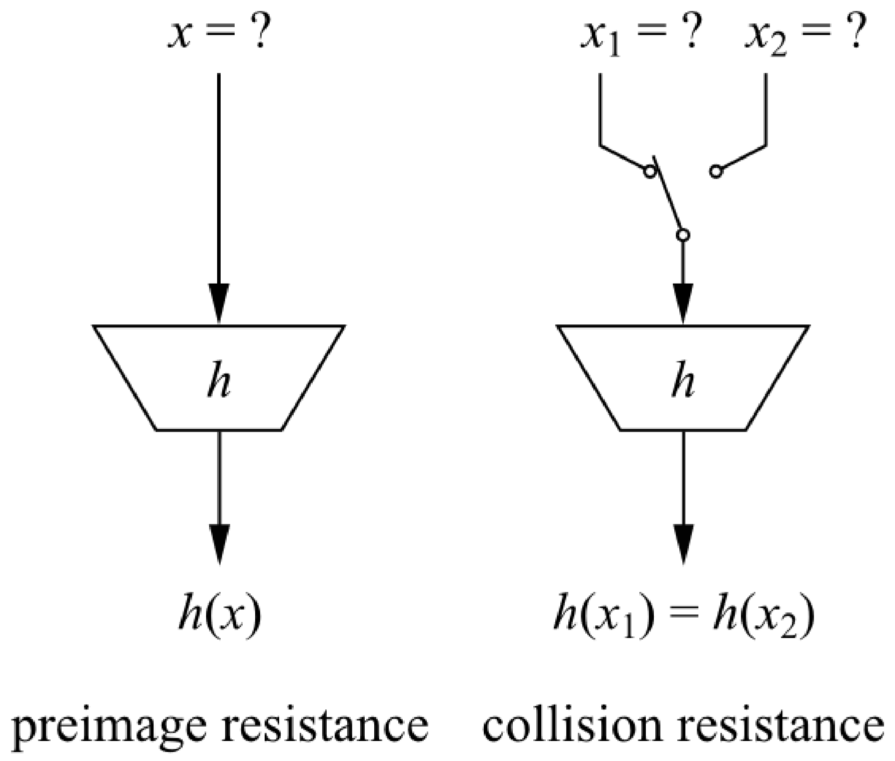
- An efficient function mapping binary strings of **arbitrary length** to binary strings of **fixed length**, called the **hash-value** or **hash-code** (also **fingerprint** or **checksum**)



Constructions for hash functions based on a block cipher are studied where the size of the hash code is equal to the block length of the block cipher and where the key size is approximately equal to the block length. A general model is presented, and it is shown that this model covers 9 schemes that have appeared in the literature. Within this general model 64 possible schemes exist, and it is shown that 12 of these are secure; they can be reduced to 2 classes based on linear transformations of variables. The properties of these 12 schemes with respect to weaknesses of the underlying block cipher are studied. The same approach can be extended to study keyed hash functions (MACs) based on block ciphers and hash functions

Cryptographic Hash Functions 密碼雜湊函數

- *H* is a function with **one-way property (pre-image resistance)** if given any *y*, it is *computationally infeasible* to find any value *x* in the domain of *H* such that *H*(*x*) = *y*
- *H* is **collision free (resistant)** if it is *computationally infeasible* to find $x' \neq x$ such that *H*(*x'*) = *H*(*x*)
- *H* is a **cryptographic hash function** if
  - *H* has one-way property
  - *H* is collision free



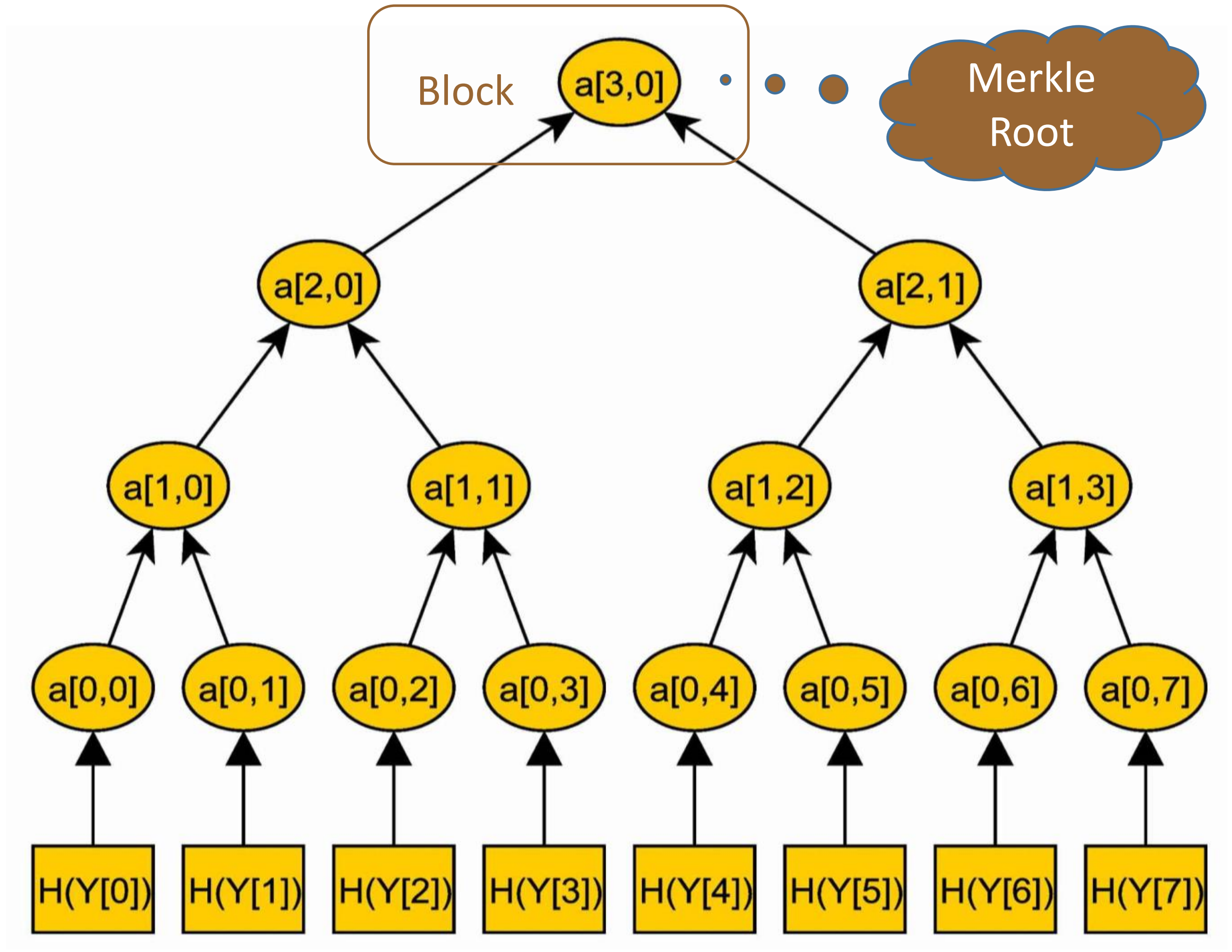
SHA: Secure Hash Algorithm

- Cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS)

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Bitwise operations	Security (bits)
SHA-1	**FIPS 180**	160	160	512	80	and, or, add, xor, rot	Theoretical attack (2^{61})
SHA-2 **FIPS 180**	SHA-224	224	256 (8×32)	512	64	and, or, xor, shr, rot, add	112
	SHA-256 Bitcoin	256					128
	SHA-384	384	512 (8×64)	1024	80	and, or, xor, shr, rot, add	192
	SHA-512	512					256
	SHA-512/224	224					112
	SHA-512/256	256					128
SHA-3 **FIPS 202**	SHA3-224	224	1600 ($5 \times 5 \times 64$)	1152	24	and, xor, rot, not	112
	SHA3-256 Ethereum (Keccak 256)	256		1088			128
	SHA3-384	384		832			192
	SHA3-512	512		576			256

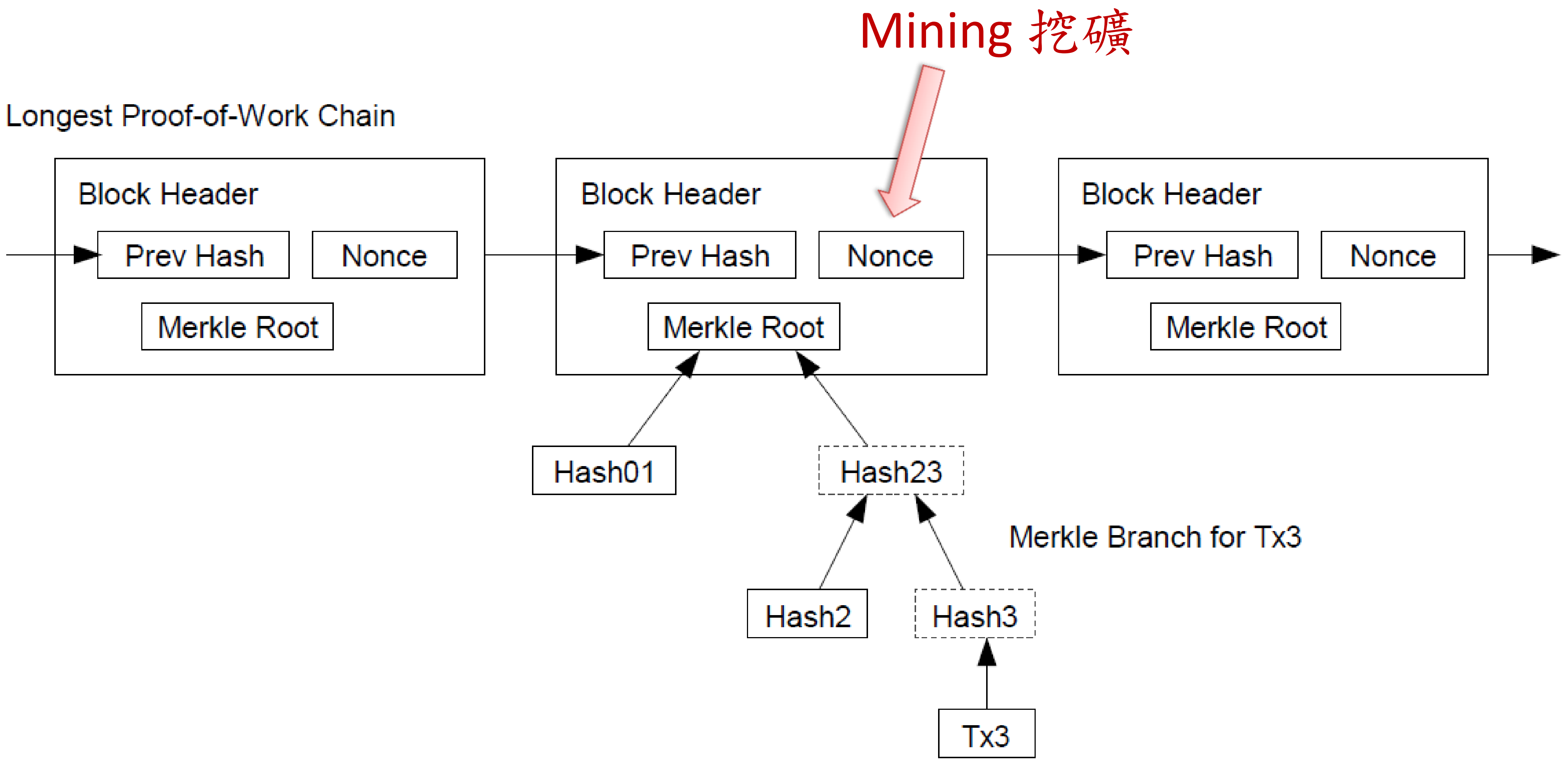
https://en.wikipedia.org/wiki/Secure_Hash_Algorithm

Merkle Tree / Hash Tree



http://commons.wikimedia.org/wiki/File:MerkleTree1.jpg

Block Chain



http://bitcoin.org/bitcoin.pdf 中本聰

Cryptowise Security

ECDSA: Choice of Two Curves

- Secp256k1 (Bitcoin and Ethereum)

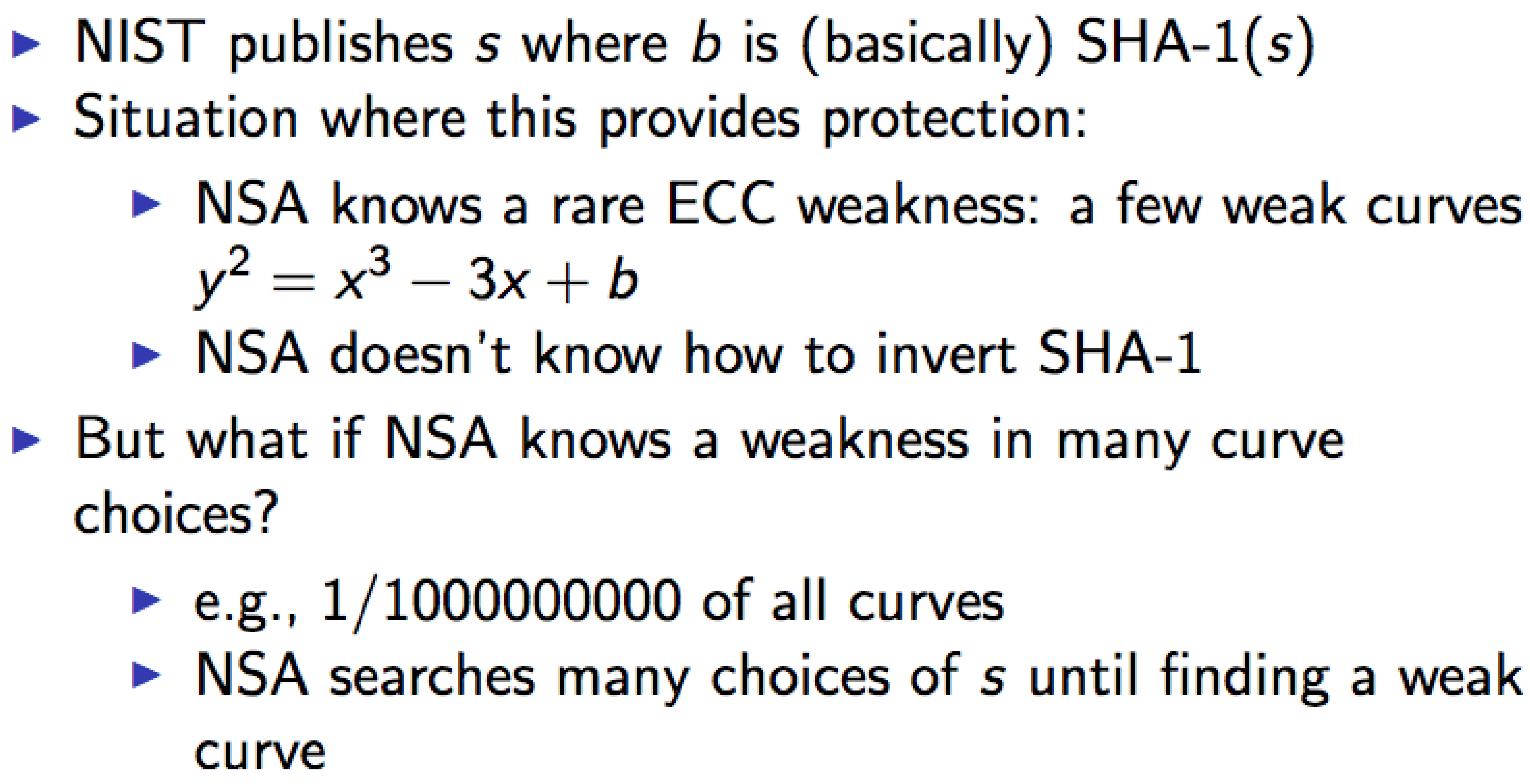
```
p = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
a = 0
b = 7
Gx = 0x79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
Gy = 0x483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8
n = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
```

- Secp256r1 (NIST P-256; parameters chosen by NSA)

```
p = 0xFFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF
a = -3
b = 0x5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
Gx = 0x6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
Gy = 0x4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5
n = 0xFFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
```

Source: http://blog.enuma.io/update/2016/11/01/a-tale-of-two-curves-hardware-signing-for-ethereum.html

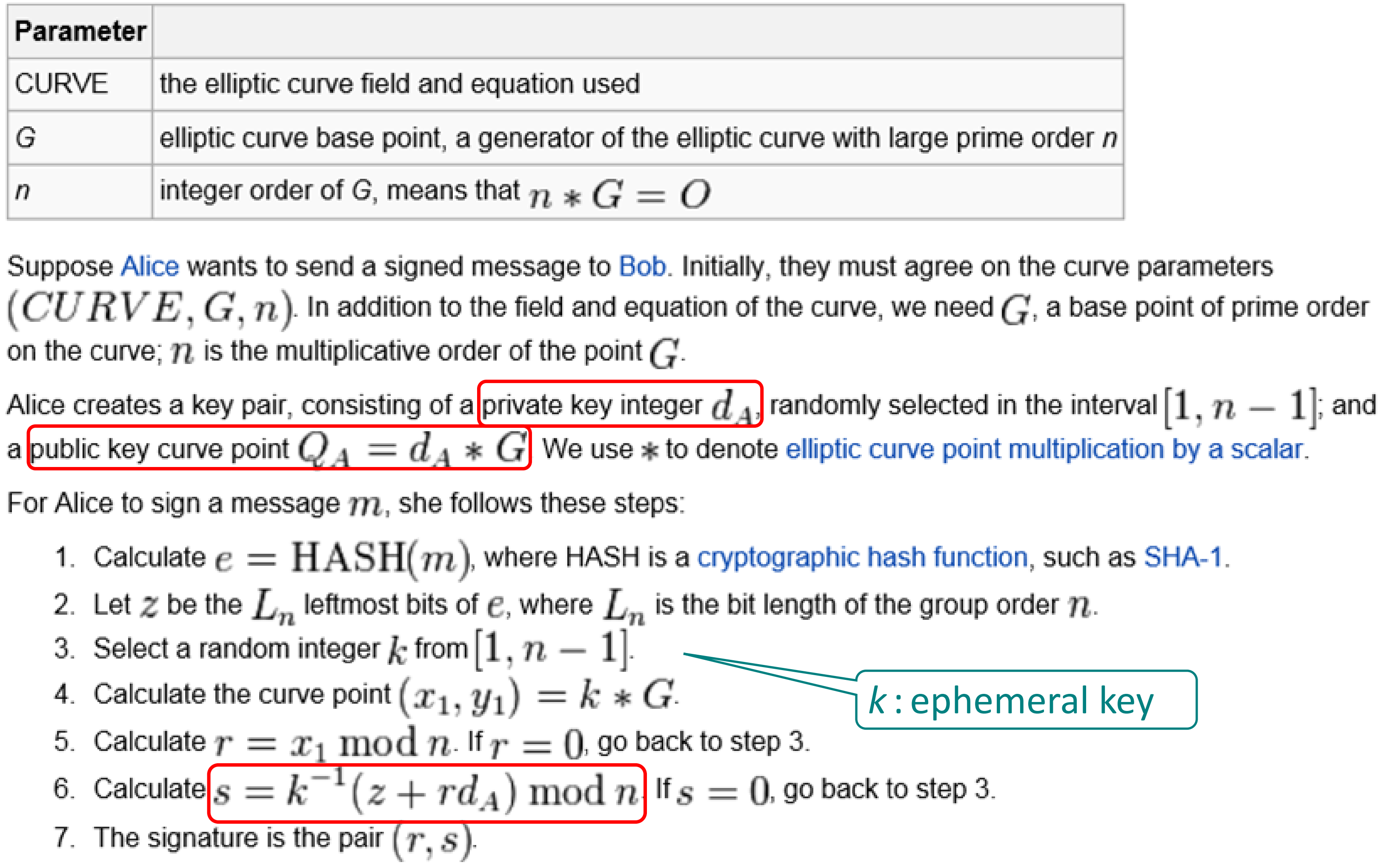
Possible Back Doors (per IEEE P1363)



- NIST publishes s where b is (basically) SHA-1(s)
- Situation where this provides protection:
  - NSA knows a rare ECC weakness: a few weak curves $y^2 = x^3 - 3x + b$
  - NSA doesn't know how to invert SHA-1
- But what if NSA knows a weakness in many curve choices?
  - e.g., 1/1000000000 of all curves
  - NSA searches many choices of s until finding a weak curve

Source: Bernstein, Daniel J., Lange, Tanja, “Security Dangers of the NIST Curves.”

ECDSA Signing 簽章



Parameter	
CURVE	the elliptic curve field and equation used
G	elliptic curve base point, a generator of the elliptic curve with large prime order *n*
n	integer order of *G*, means that $n * G = O$

Suppose Alice wants to send a signed message to Bob. Initially, they must agree on the curve parameters $(CURVE, G, n)$. In addition to the field and equation of the curve, we need G, a base point of prime order on the curve; n is the multiplicative order of the point G.

Alice creates a key pair, consisting of a private key integer d_A, randomly selected in the interval $[1, n-1]$; and a public key curve point $Q_A = d_A * G$. We use $*$ to denote elliptic curve point multiplication by a scalar.

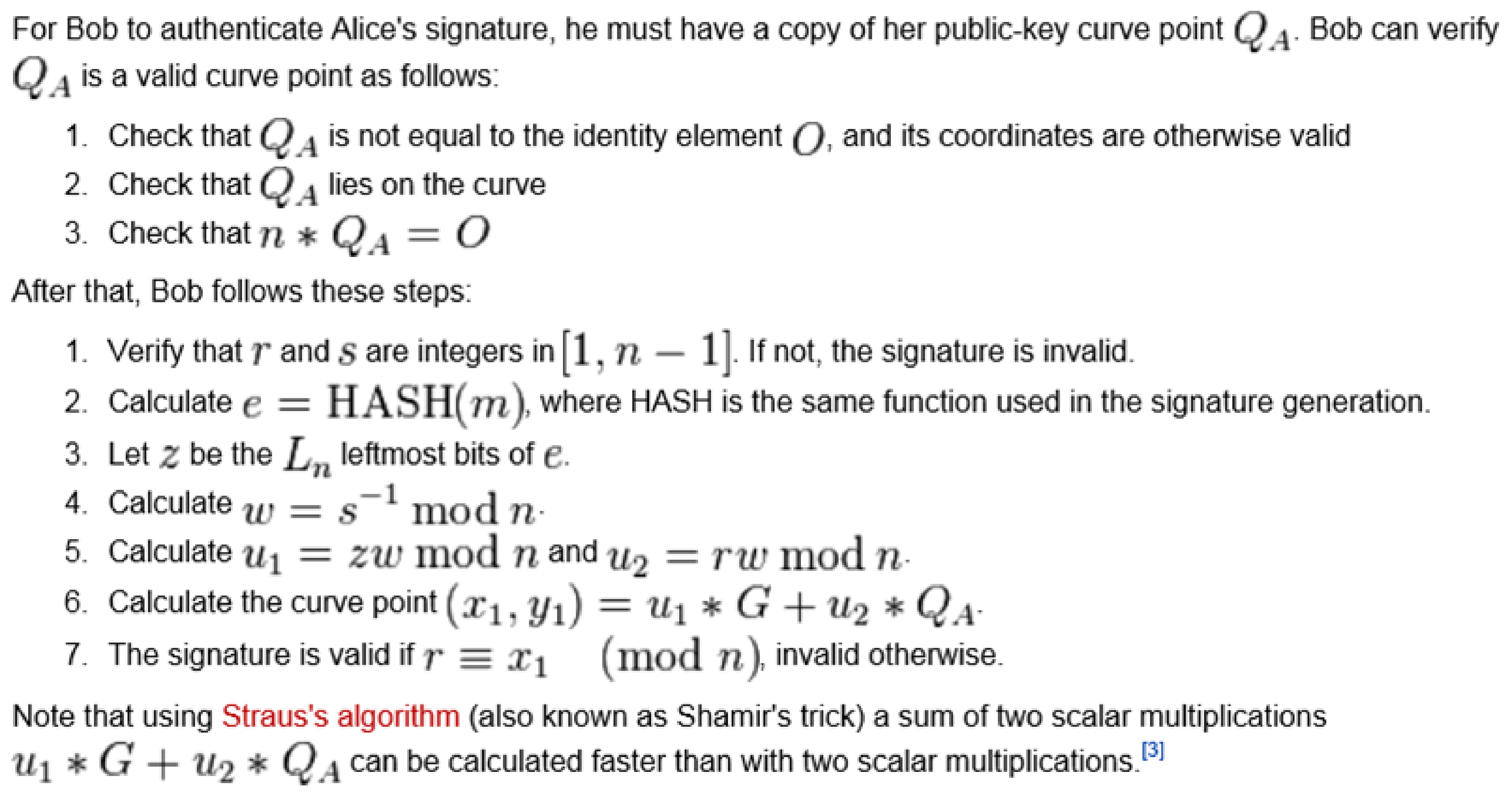
For Alice to sign a message m, she follows these steps:

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-1.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a random integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k * G$.
5. Calculate $r = x_1 \bmod n$. If $r = 0$, go back to step 3.
6. Calculate $\boxed{s = k^{-1}(z + rd_A) \bmod n}$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

k : ephemeral key

http://en.wikipedia.org/wiki/Elliptic_Curve_DSA

ECDSA Verification 驗章



For Bob to authenticate Alice's signature, he must have a copy of her public-key curve point Q_A. Bob can verify Q_A is a valid curve point as follows:

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid
2. Check that Q_A lies on the curve
3. Check that $n * Q_A = O$

After that, Bob follows these steps:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \bmod n$.
5. Calculate $u_1 = zw \bmod n$ and $u_2 = rw \bmod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 * G + u_2 * Q_A$.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

Note that using Straus's algorithm (also known as Shamir's trick) a sum of two scalar multiplications $u_1 * G + u_2 * Q_A$ can be calculated faster than with two scalar multiplications.[3]

http://en.wikipedia.org/wiki/Elliptic_Curve_DSA

Ephemeral Key & RNG

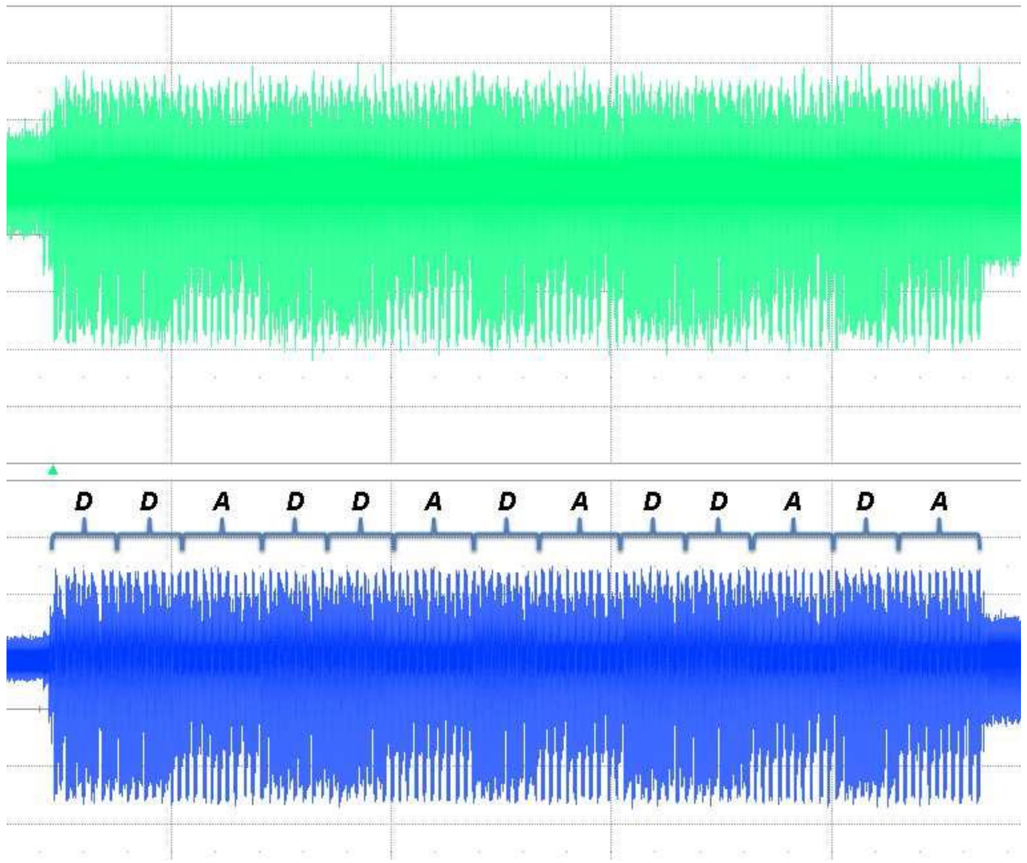
- The **entropy**, **secrecy**, and **uniqueness** of the DSA / ECDSA **random ephemeral key** ***k*** is critical
  - Violating any one of the above three requirements can reveal the entire private key to an attacker
  - Using the same value twice (even while keeping *k* secret), using a predictable value, or leaking even a few bits of *k* in each of several signatures, is enough to break DSA / ECDSA
- [December 2010] The ECDSA private key used by **Sony** to sign software for the **PlayStation 3** game console was recovered, because Sony implemented *k* as static instead of random

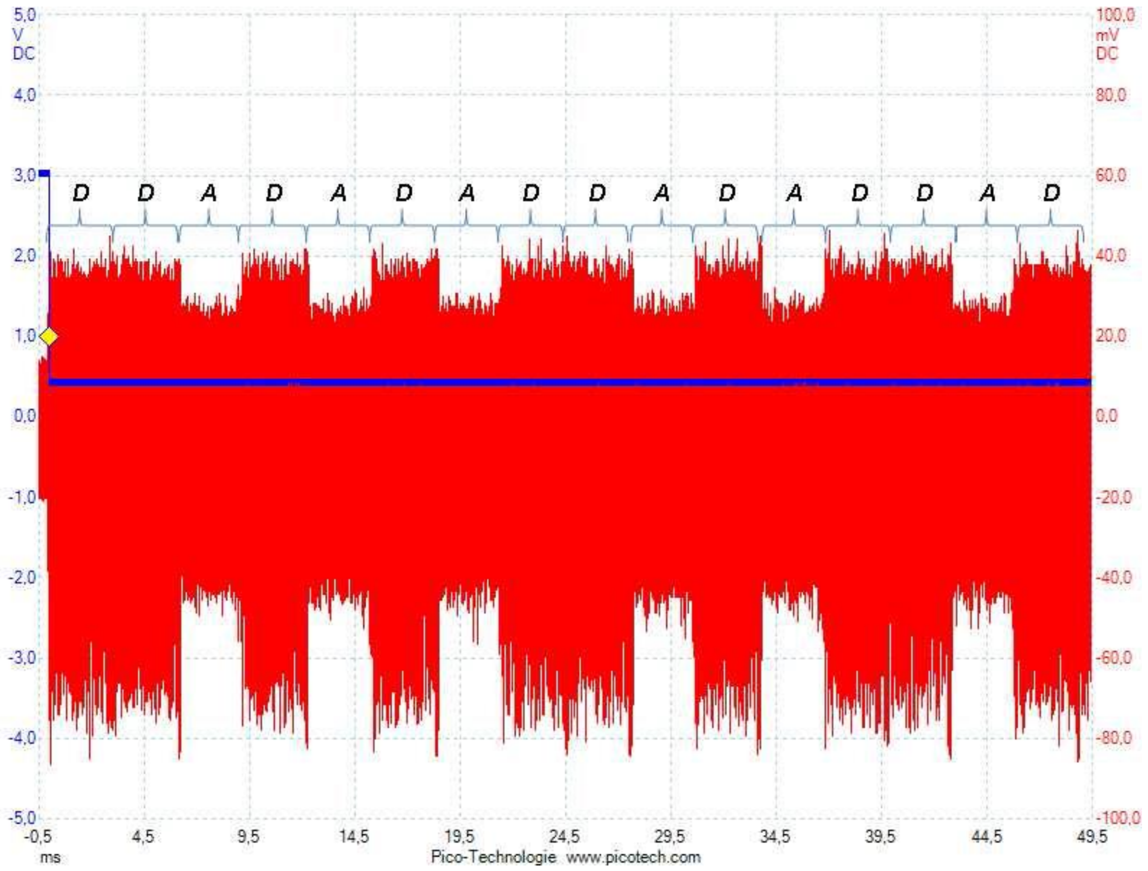
Ephemeral Key & RNG

- [August 2013] Bugs in some implementations of the Java class *SecureRandom* sometimes generated collisions in *k*, allowing in stealing **bitcoins** from the containing wallet on **Android app**
- [August 2013] 158 accounts had used the same signature nonces *r* value in more than one signature. The total remaining balance across all 158 accounts is only 0.00031217 BTC. The address, 1HKywxiL4JziqXrzLKhmB6a74ma6kxbSDj, appears to have stolen bitcoins from 10 of these addresses. This account made 11 transactions between March and October 2013. These transactions have netted this account over 59 bitcoins.
- This issue can be prevented by deriving *k* deterministically from the **private key** and the **message hash**, as described by **RFC 6979**

http://www.theregister.co.uk/2013/08/12/android_bug_batters_bitcoin_wallets http://eprint.iacr.org/2013/734.pdf

Side-Channel Attacks 旁通道攻擊



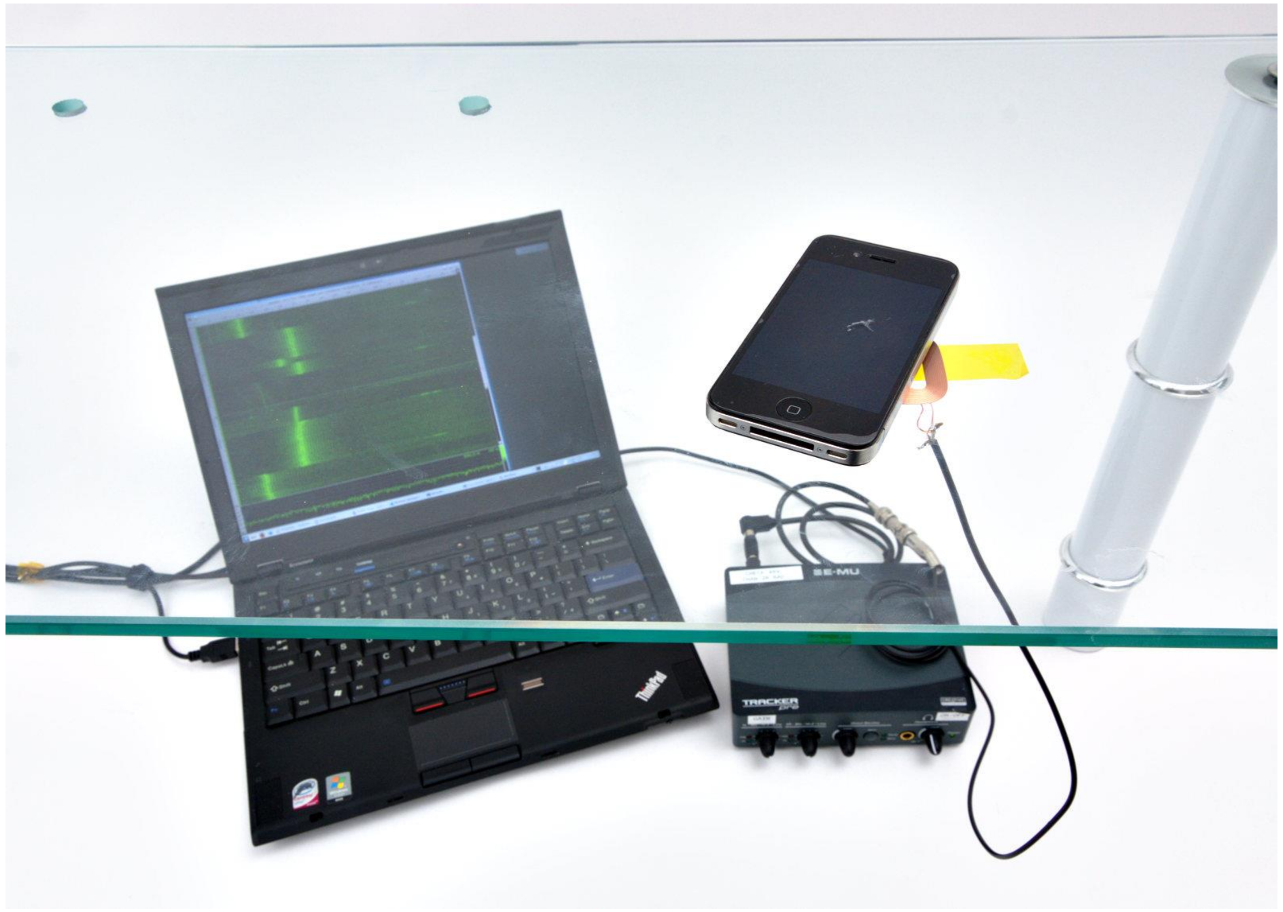


D (double) or **A** (add) depends on the bits of **Private Key**

Image Courtesy https://eprint.iacr.org/2015/354.pdf

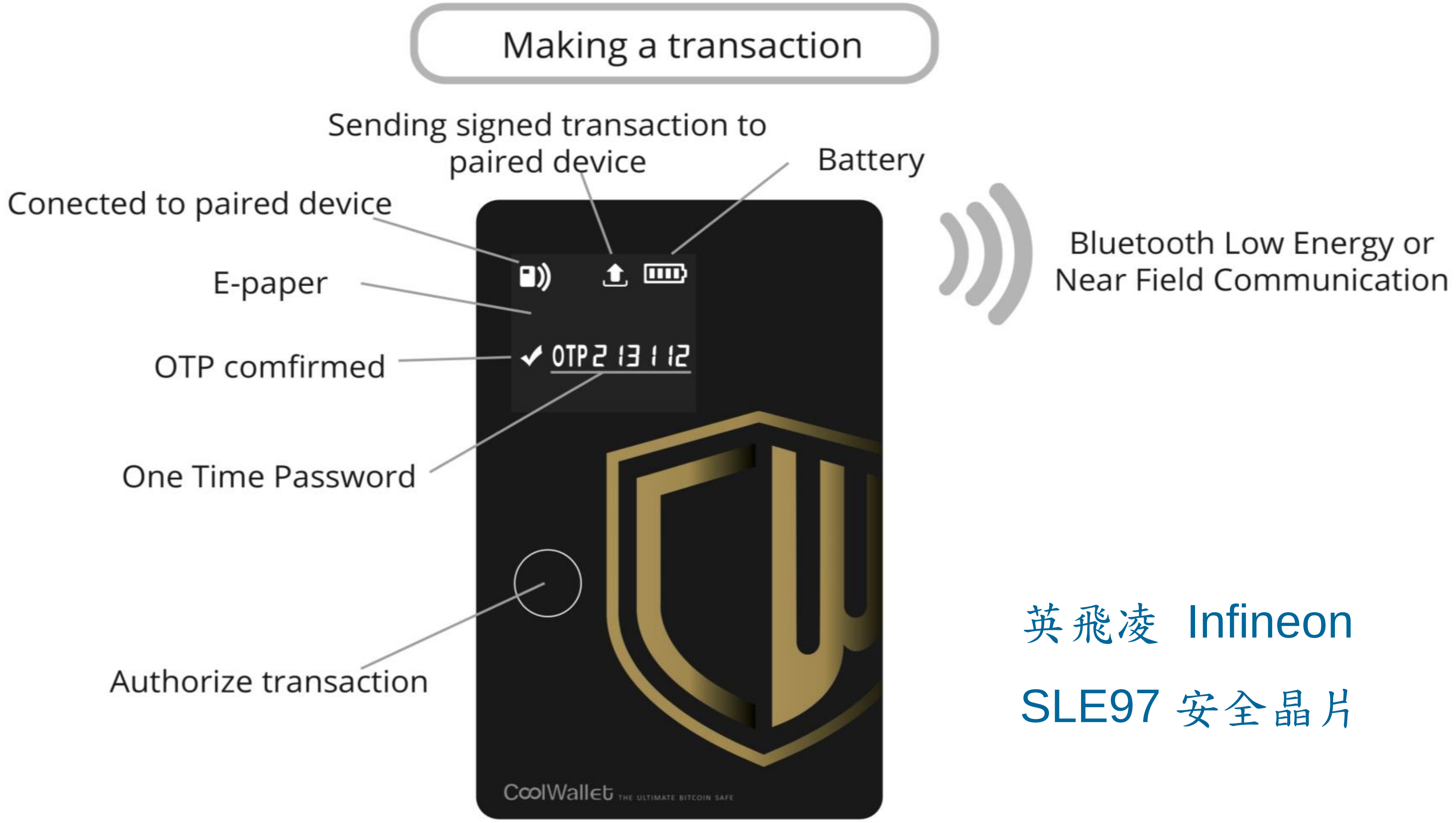
ECDSA Key Extraction from Mobile Devices

Fully extract secret signing keys from OpenSSL and CoreBitcoin running on iOS devices.



Sourse: https://www.tau.ac.il/~tromer/mobilesc

CoolWallet for Hardware Signing


Making a transaction
Sending signed transaction to paired device
Battery
Conected to paired device
E-paper
OTP comfirmed
One Time Password
Authorize transaction
Bluetooth Low Energy or Near Field Communication
OTP 2 13 1 12
CoolWallet THE ULTIMATE BITCOIN SAFE

英飛凌 Infineon

SLE97 安全晶片

Quantum Resistant Suite

- In August, 2015, NSA announced that it is planning to transition "in the not too distant future" to a new cipher suite that is resistant to quantum attacks.
- NSA advised: "For those partners and vendors that have not yet made the transition to Suite B algorithms, we recommend not making a significant expenditure to do so at this point but instead to prepare for the upcoming quantum resistant algorithm transition.“
- Prediction: Post-Quantum blockchains are appearing soon

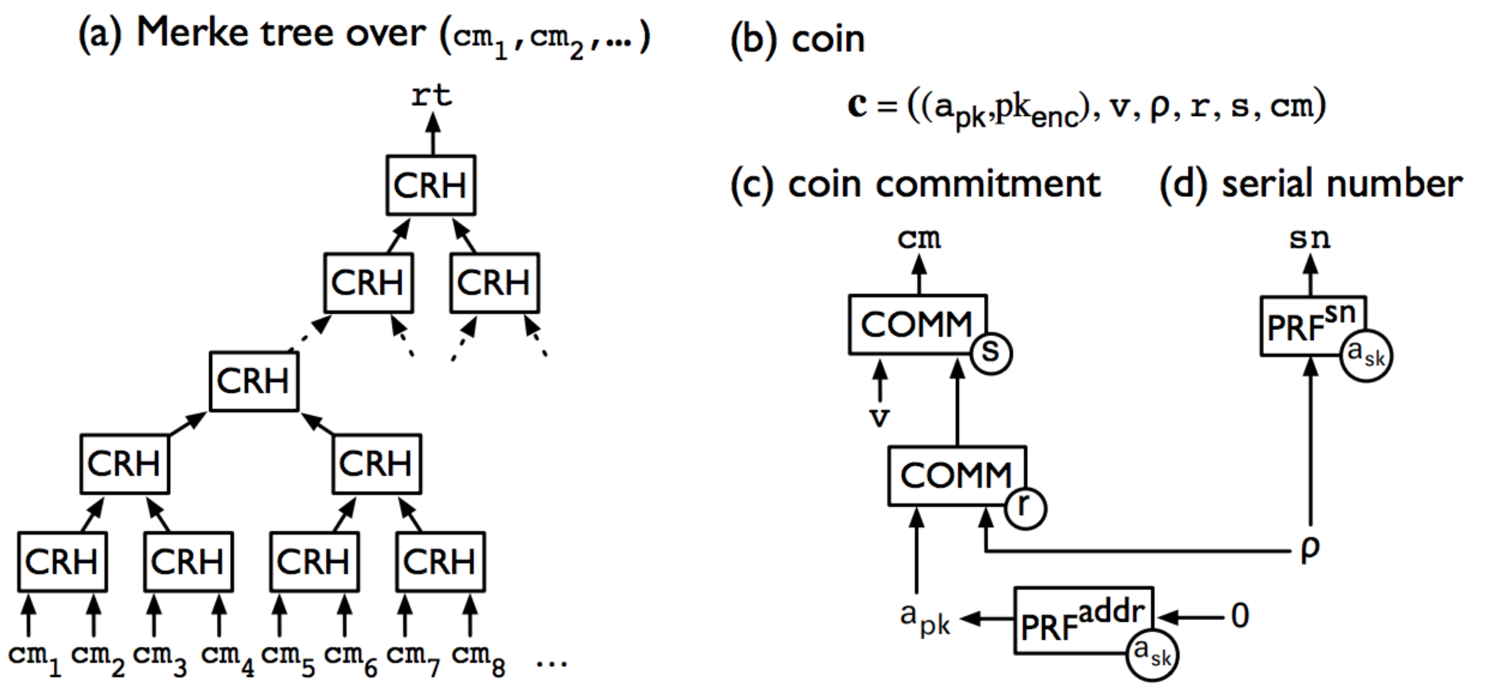
https://en.wikipedia.org/wiki/NSA_Suite_B_Cryptography

Collision Resistance of SHA-2, -3 Hash Functions

- Blockchains depend on collision-resistant hash functions such as SHA-2 and SHA-3 for consensus (proof of work), wallet generation, and transaction signing. A successful pre-image attack would be a serious problem.
- What is the chance of a successful pre-image attack on SHA-2 and SHA-3 with the help of quantum computation?
- Attacks on both functions require on the order of 2^{128} queries in a quantum block-box model, hence suggesting than an attack is 275 billion times more expensive than a simple query analysis would suggest.

Source: Amy, Di Matteo, Gheorghiu, et. al., “Estimating the Cost of Generic Quantum Pre-Image Attacks on SHA-2 and SHA-3.”

Zero-Knowledge Proofs for Blockchain Privacy


(a) Merke tree over $(cm_1, cm_2, \dots)$
rt
CRH
cm_1 cm_2 cm_3 cm_4 cm_5 cm_6 cm_7 cm_8 …
(b) coin
$c = ((a_{pk}, pk_{enc}), v, \rho, r, s, cm)$
(c) coin commitment
cm
COMM
s
v
r
a_{pk}
PRF^{addr}
a_{sk}
0
(d) serial number
sn
PRF^{sn}
a_{sk}
ρ

rt	=	*Merkle-tree root*
cm	=	*coin commitment*
sn	=	*serial number*
v	=	*coin value*
r, s	=	*commitment rand.*
ρ	=	*serial number rand.*
(a_{pk}, pk_{enc})	=	*address public key*
(a_{sk}, sk_{enc})	=	*address secret key*

Fig. 1: **(a)** Illustration of the CRH-based Merkle tree over the list CMList of coin commitments. **(b)** A coin **c**. **(c)** Illustration of the structure of a coin commitment cm. **(d)** Illustration of the structure of a coin serial number sn.

Source: Ben-Sasson, Chiesa, Garman, et. al., “Zerocash: Decentralized Anonymous Payment from Bitcoin.”

Non-Crytpowise Security

Consensus Algorithms

- Consensus tolerating Byzantine failures must satisfy:
  - Termination – every correct process decides some value.
  - Validity – if all correct processes propose the same value *v*, then all correct processes decide *v*.
  - Integrity – if a correct process decides *v*, then *v* must have been proposed by some correct process.
  - Agreement – every correct process must agree on the same value.

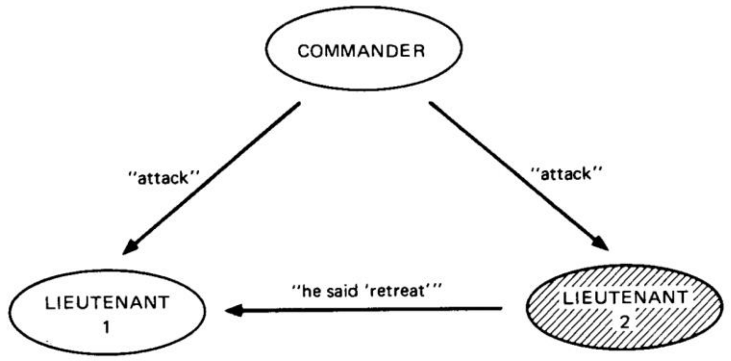


Fig. 1. Lieutenant 2 a traitor.

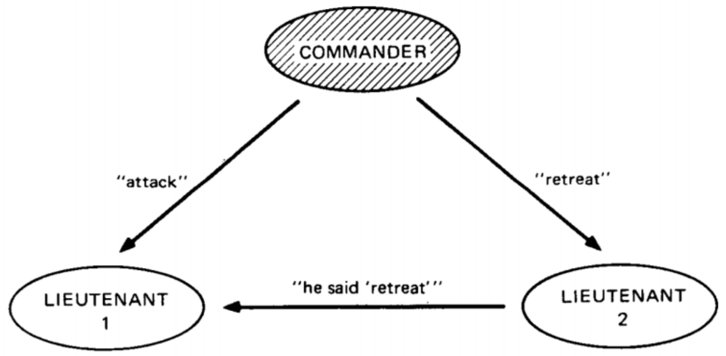


Fig. 2. The commander a traitor.

Sources: Lamport, L., Shostak, R., Pease, M., "The Byzantine Generals Problem."
Castro, M., Liskov, B., "Practical Byzantine Fault Tolerance and Proactive Recovery."

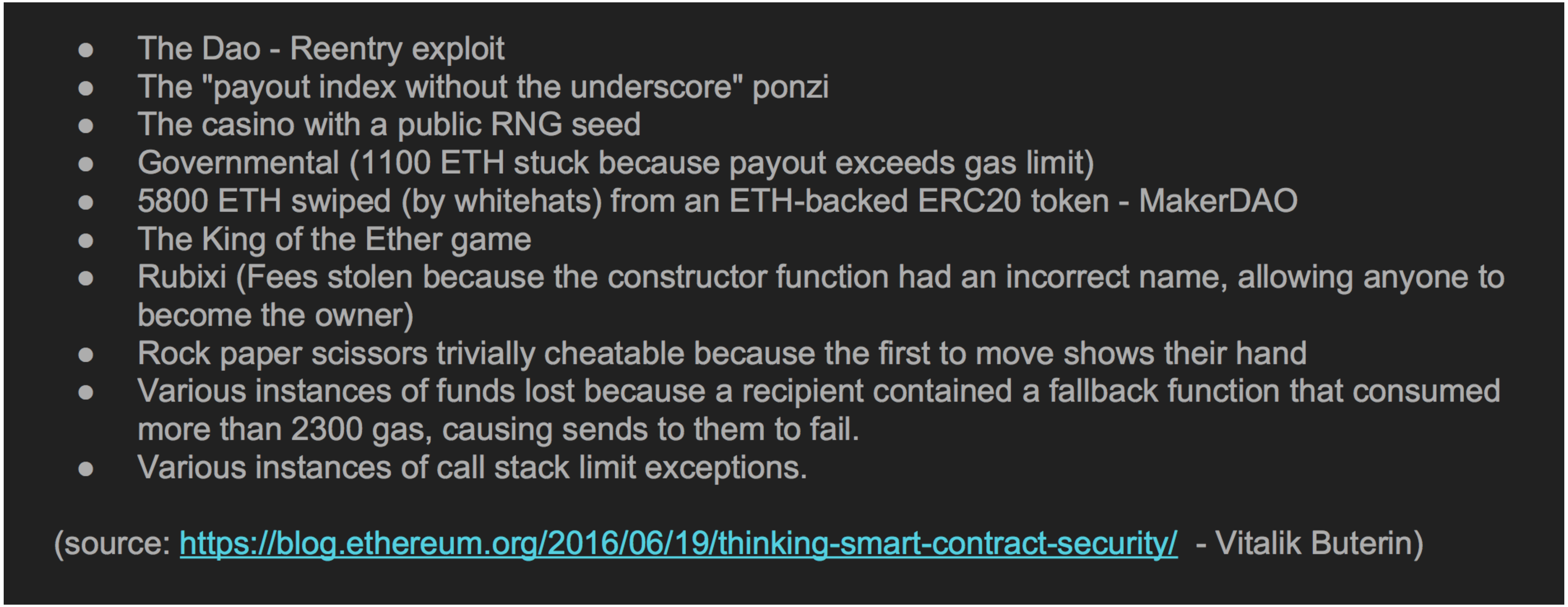
A Comparison of Consensus Algorithms

- Decentralized Control – anyone is able to participate and no central authority dictates whose approval is required for consensus.
- Low Latency – consensus can be reached in a few seconds.
- Flexible Trust – users have the freedom to trust any combination of parties they see fit.
- Asymptotic Security – safety rests on digital signatures and hash families whose parameters can be tuned to protect against adversaries with unlimited computing power.

Algorithm	Decentralized Control	Low Latency	Flexible Trust	Asymptotic Security
Proof of Work	✔			
Proof of Stake	✔	maybe		maybe
PBFT		✔	✔	✔
Tendermint	✔	✔		✔

Source: Mazieres, David, "The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus."

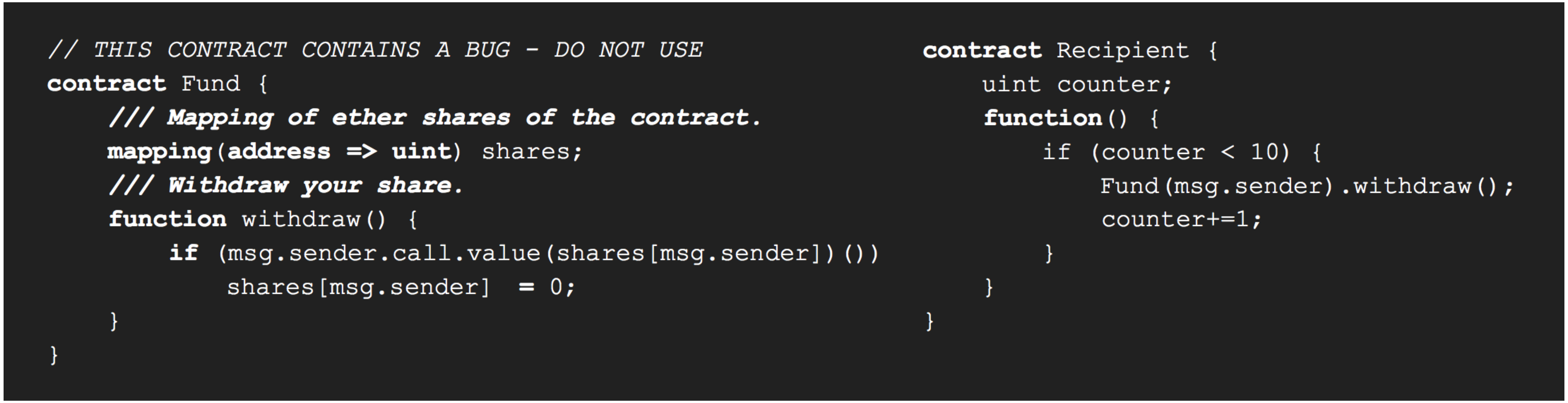
Smart Contract Failures



- The Dao - Reentry exploit
- The "payout index without the underscore" ponzi
- The casino with a public RNG seed
- Governmental (1100 ETH stuck because payout exceeds gas limit)
- 5800 ETH swiped (by whitehats) from an ETH-backed ERC20 token - MakerDAO
- The King of the Ether game
- Rubixi (Fees stolen because the constructor function had an incorrect name, allowing anyone to become the owner)
- Rock paper scissors trivially cheatable because the first to move shows their hand
- Various instances of funds lost because a recipient contained a fallback function that consumed more than 2300 gas, causing sends to them to fail.
- Various instances of call stack limit exceptions.

(source: https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/ - Vitalik Buterin)

The DAO Reentrancy Bug

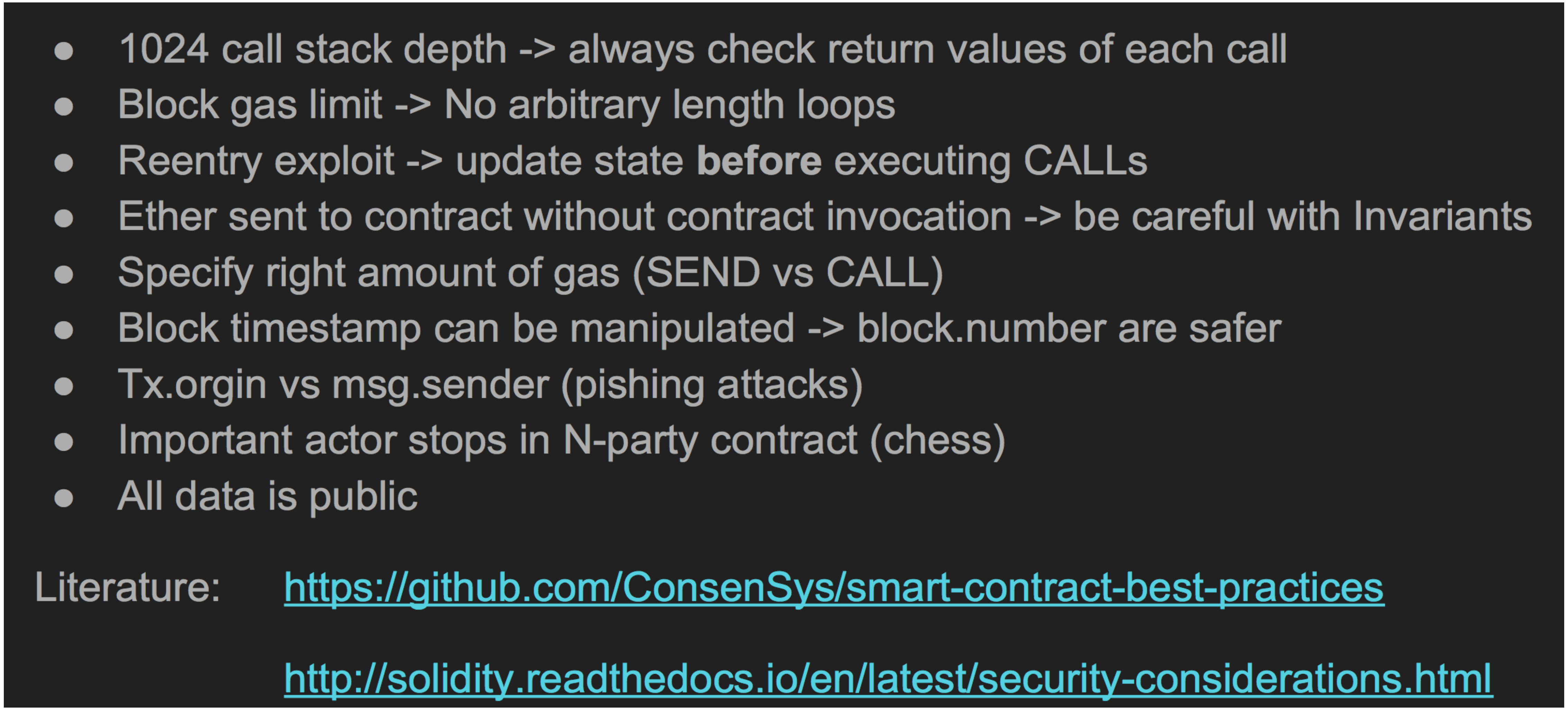


```
// THIS CONTRACT CONTAINS A BUG - DO NOT USE
contract Fund {
    /// Mapping of ether shares of the contract.
    mapping(address => uint) shares;
    /// Withdraw your share.
    function withdraw() {
        if (msg.sender.call.value(shares[msg.sender])())
            shares[msg.sender] = 0;
    }
}
```

```
contract Recipient {
    uint counter;
    function() {
        if (counter < 10) {
            Fund(msg.sender).withdraw();
            counter+=1;
        }
    }
}
```

Source: Jentzsch, Christoph, “Smart Contract Security and Decentralized Governance.”

Establishing Security Patterns

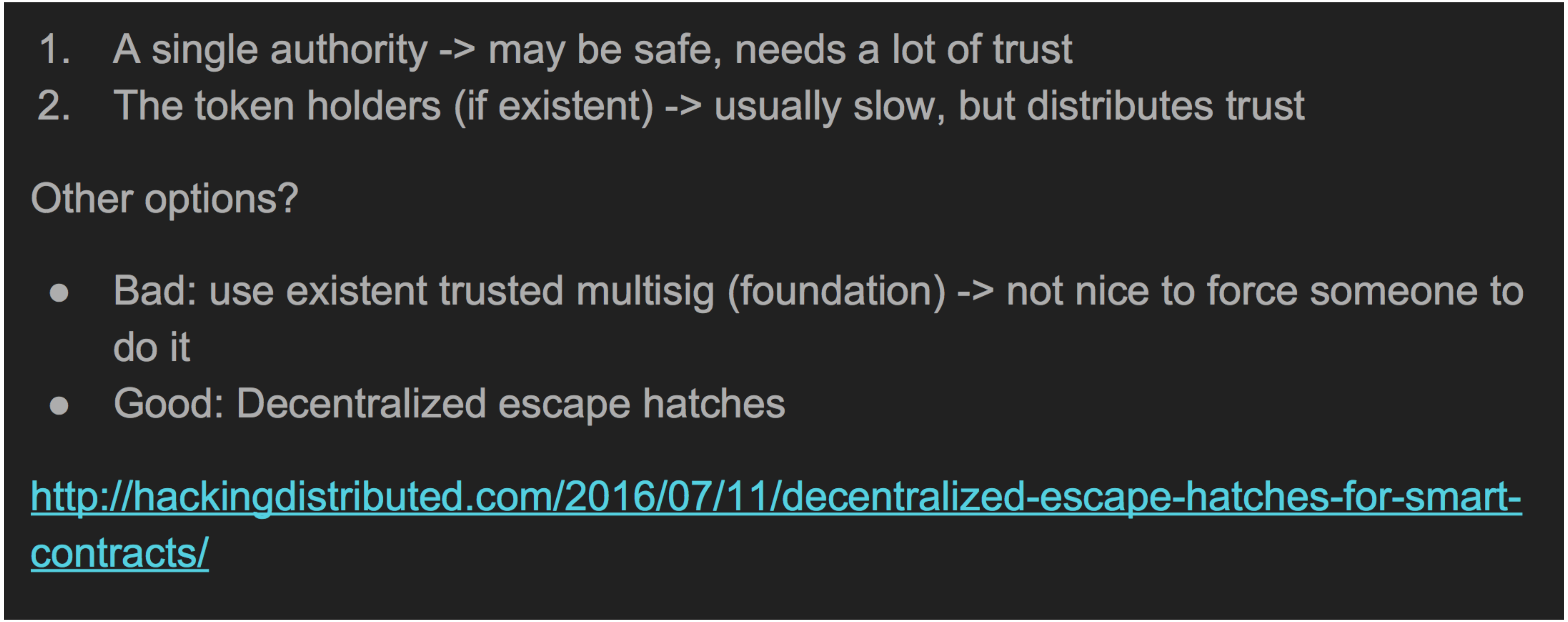


- 1024 call stack depth -> always check return values of each call
- Block gas limit -> No arbitrary length loops
- Reentry exploit -> update state **before** executing CALLs
- Ether sent to contract without contract invocation -> be careful with Invariants
- Specify right amount of gas (SEND vs CALL)
- Block timestamp can be manipulated -> block.number are safer
- Tx.orgin vs msg.sender (pishing attacks)
- Important actor stops in N-party contract (chess)
- All data is public

Literature: https://github.com/ConsenSys/smart-contract-best-practices

http://solidity.readthedocs.io/en/latest/security-considerations.html

Smart Contract Governance



1. A single authority -> may be safe, needs a lot of trust
2. The token holders (if existent) -> usually slow, but distributes trust

Other options?

- Bad: use existent trusted multisig (foundation) -> not nice to force someone to do it
- Good: Decentralized escape hatches

http://hackingdistributed.com/2016/07/11/decentralized-escape-hatches-for-smart-contracts/

Smart Contract Security Conclusions

- Practice prudent design (invariants, coverage, formal verification)
- Defense in depth (cap transaction amount, time delays, circuit breakers)
- Design escape hatches (updateable contracts, multisig rescue)

→ Keep smart contracts simple (only decentralize what absolutely needs to be decentralized). We are still in the early days.

Miscellaneous Blockchain Exploits

- DAO Reentrancy Bug (>$60 million loss) – mitigated by hard fork, time delays
- Bitfinex Compromise (>$60 million loss) – advanced persistent threat
- Mt. Gox (>$400 million loss) – insider incompetence/fraud
- Bitstamp ($5 million loss) – social engineering
- Bitcoinica ($2 million loss) – insider incompetence/fraud
- Many others totaling over $1 billion in losses

Thank you!

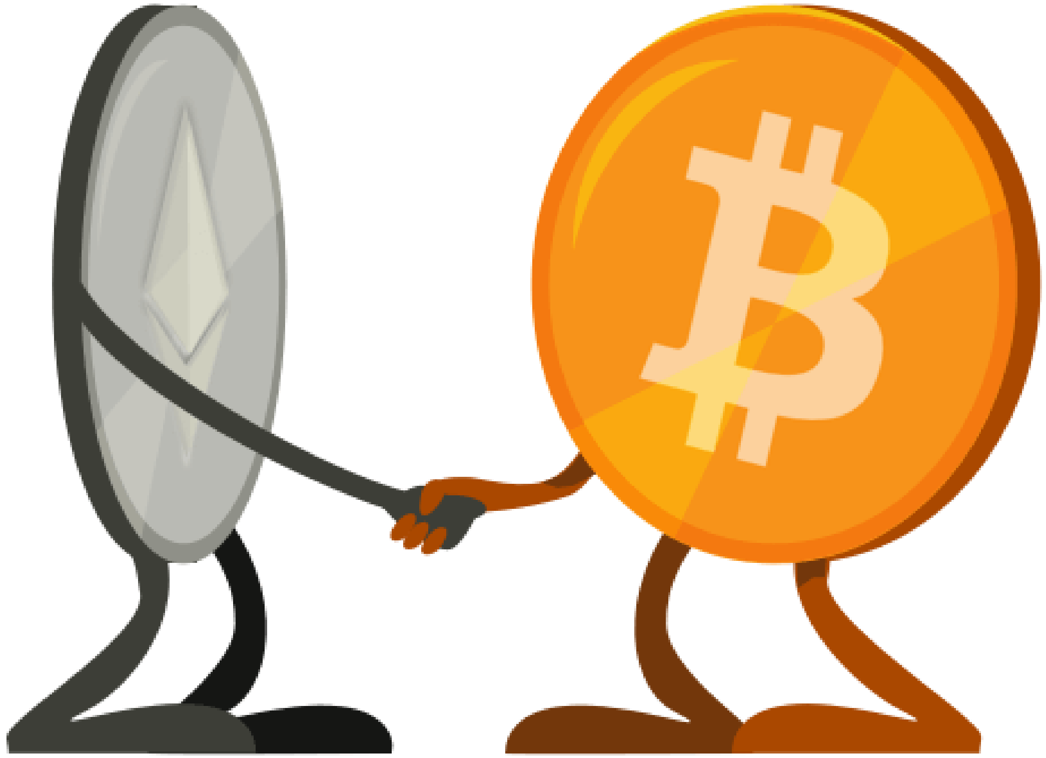


Image Courtesy https://www.ethereum.org/ether