

Security and Privacy Engineering

Introduction

Aljosha Judmayer

aljosha.judmayer@univie.ac.at

GPG FP:

4016 59DB D4E0 A908 FCDF
7BFB A40D 40AC E687 AADD

Nicholas Stifter

nicholas.stifter@univie.ac.at

GPG FP:

10C6 4FD1 19B1 B399 4A2B
6D7B 5EB9 556A 4339 97A9

Organisational Details

- Lectures (best effort recording if possible)
 - Slides will be uploaded to Moodle
- Grading:
 - Written Exam (40%)
 - Assignment (60%)
 - Challenges
 - Security Incident Reports
- Recommended book(s) are intended as supplementary material not mandatory
- Lecture attendance is not strictly mandatory, however we encourage an interactive and open discussion - You have the chance to ask us anything, so make use of it ;)



Books for further study

- Security Engineering, Ross Anderson
 - <https://www.cl.cam.ac.uk/~rja14/book.html> (2nd edition open access)
- Background and reference regarding cryptography:
 - A Graduate Course in Applied Cryptography, Dan Boneh and Victor Shoup
 - <https://toc.cryptobook.us/> (open access)
 - Introduction to Modern Cryptography, Jonathan Katz and Yehuda Lindell
 - <https://www.cs.umd.edu/~ikatz/imc.html>
 - A Computational Introduction to Number Theory and Algebra, Victor Shoup
 - <https://shoup.net/ntb/> (open access)
 - Handbook of Applied Cryptography, Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone
 - <https://cacr.uwaterloo.ca/hac/> (open access)
- Background and reference regarding distributed systems:
 - Distributed Algorithms, Nancy Lynch
 - <https://groups.csail.mit.edu/tds/distalgs.html>
 - Introduction to Reliable and Secure Distributed Programming, Christian Cachin, Rachid Guerraoui, Luís Rodrigues
 - <https://www.distributedprogramming.net/>



Planned Course Structure

- Introduction to Security Engineering
- Fundamentals of applied cryptography
- Fundamentals of Distributed Systems
- Case studies
- Introduction to Blockchains and DLT
- Introduction to Smart Contracts & Ethereum

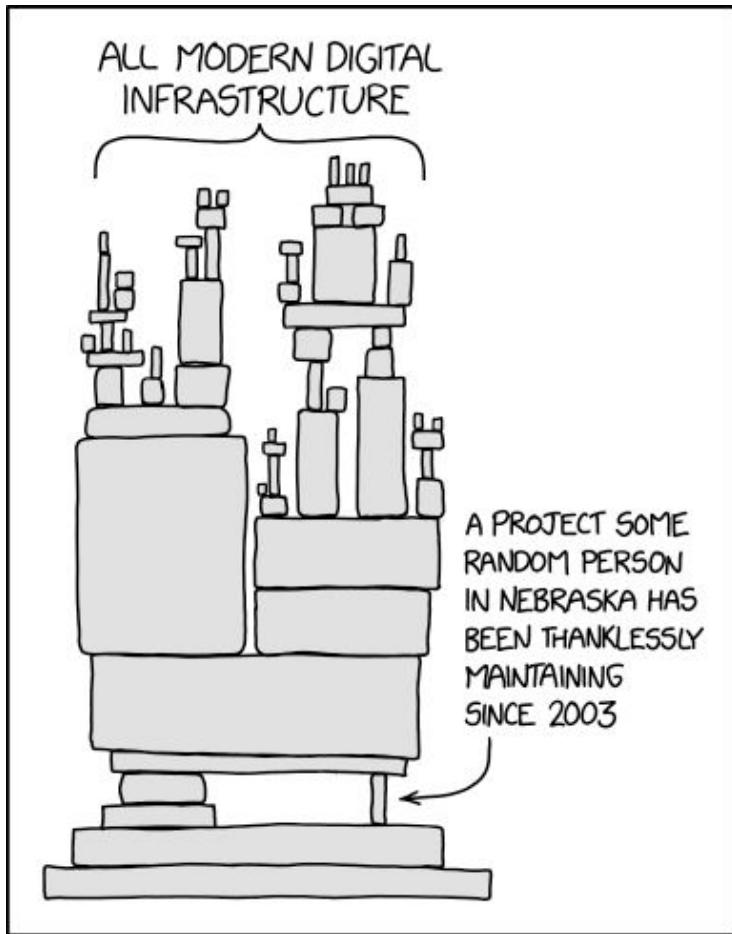


Security Engineering



universität
wien

SBA
Research



[1] <https://imgs.xkcd.com/comics/dependency.png>

Security Engineering

*“Security engineering is about building systems to remain dependable in the face of **malice, error, or mischance**. As a discipline, it focuses on the tools, processes, and methods needed to design, implement, and test **complete systems**, and to adapt existing systems as their environment evolves.*

*Security engineering requires **cross-disciplinary expertise**, ranging from cryptography and computer security through hardware tamper-resistance to a knowledge of economics, applied psychology, organisations and the law.” [1]*

[1] Ross Anderson, Security Engineering, 3rd Edition

What is a secure system?

- *There is no “100% secure system”!*
 - Not every possible issue from hardware-/software failures to bribing and social engineering can be mitigated
 - For practical cryptographic schemes, there still is an astronomically small chance that an attacker will be successful
- *Answer depends on your goals and threat model!*

A secure system achieves its **security goals** under a certain **threat-/attack model**

What is a secure system?

- *The goal of most (IT) security measures is it to **increase the costs for an attacker** such that attacks are no longer viable*
 - Costs can be measured in resources such as computational steps required or monetary expenses
- *Asymmetric costs/effort*
 - Security measure should be low cost for users
 - Security measure should result in high costs for attackers



What is a system?

- In practice a *system* can be [1]:
 - Product or component such as a cryptographic protocol, smartcard, hardware, laptop, server, mobile phone, ...
 - One or more of the above plus operating system, communication infrastructure, ...
 - The above plus one or more applications (browser, banking app, ...)
 - Any or all of the above plus IT staff
 - Any or all of the above plus internal users and management
 - Any or all of the above plus customers and other external users ...
 - ...
- Be sure to use the **same terminology** as your communication partner
- Agreed upon **definitions** are important

[1] Ross Anderson, Security Engineering, 3rd Edition



The Formal Approach: Principles of modern Cryptography [1]

- **Formal Definitions**

“If you don’t understand what you want to achieve, how can you possibly know when (or if) you have achieved it?” [1]

- What threats are in scope, what security guarantees are desired
 - **security goals** (what should be achieved)
 - E.g., in context of encryption: *a ciphertext should leak no additional information about the underlying plaintext*
 - **threat model** (power of the attacker)
 - E.g., in context of encryption: *ciphertext-only, known-plaintext, chosen-plaintext, chosen-ciphertext, ...*
- There are multiple valid ways to define what “secure” (or privacy preserving) means in the respective context. Designers have to think about what is essential.

- **Precise Assumptions**

- Assumptions are statements that are not proven but are instead conjectured to be true
 - assumption about the hardness of a mathematical problem
 - most of cryptography requires the assumption $P \neq NP$

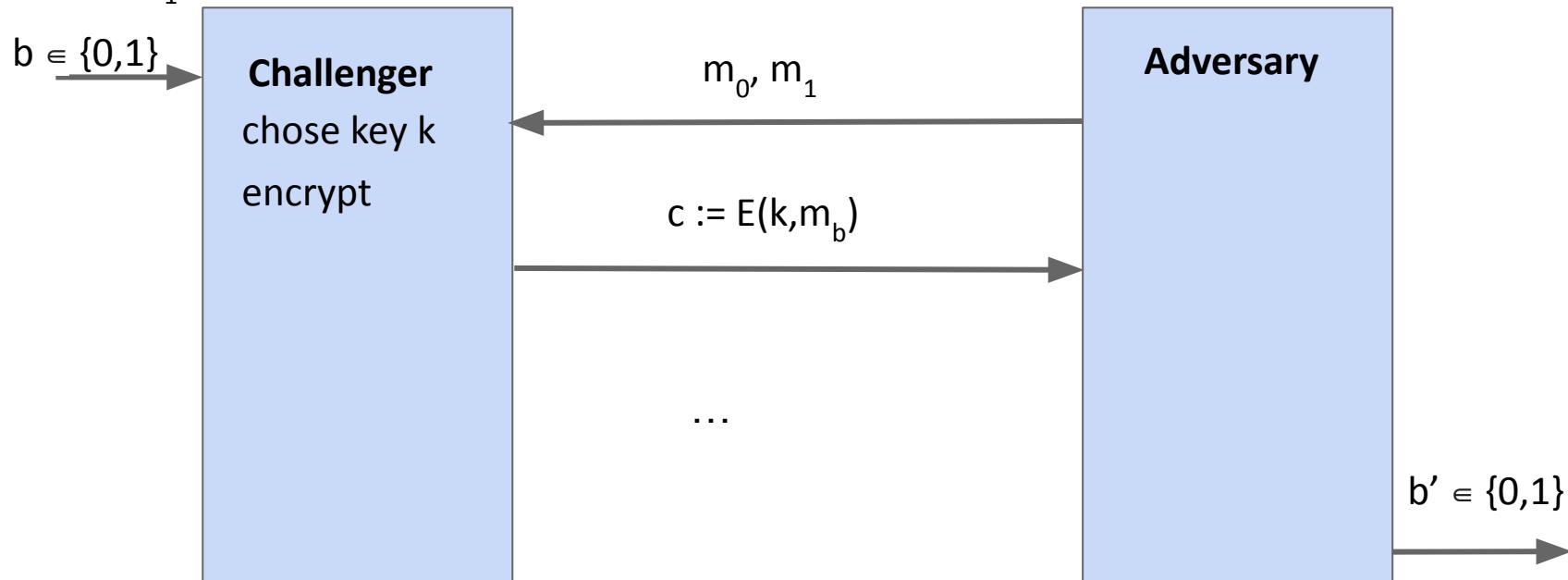
- **Proofs of Security**

- Proof that a construction satisfies a given *definition* under certain *assumptions*

[1] Katz and Lindell, Introduction to Modern Cryptography, 3rd Edition

The Formal Approach: Example of a Game Based Proof

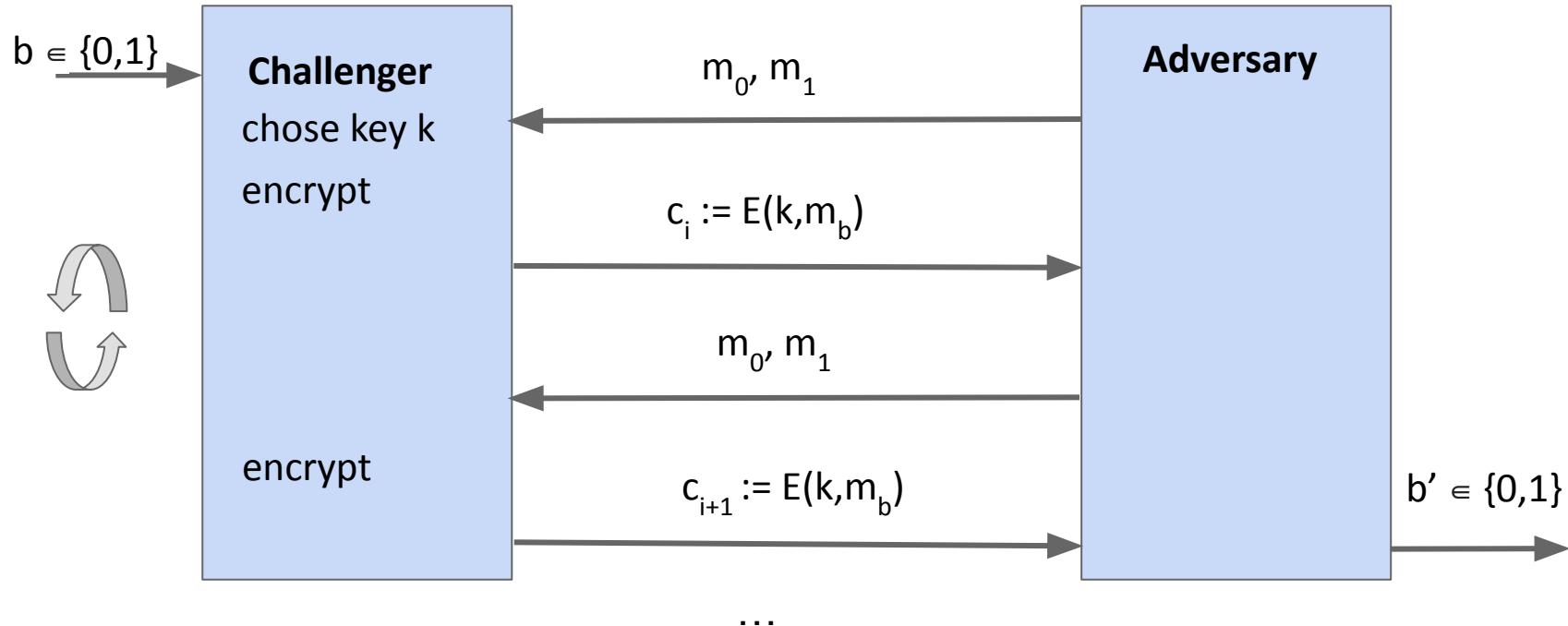
Adversary, sends two (equally long) messages, must find out if in experiment 0 , or experiment 1. In EXP 0 always gets encryption of m_0 , in EXP 1 always gets encryption of m_1



The Formal Approach: Example of a Game Based Proof

Semantic security

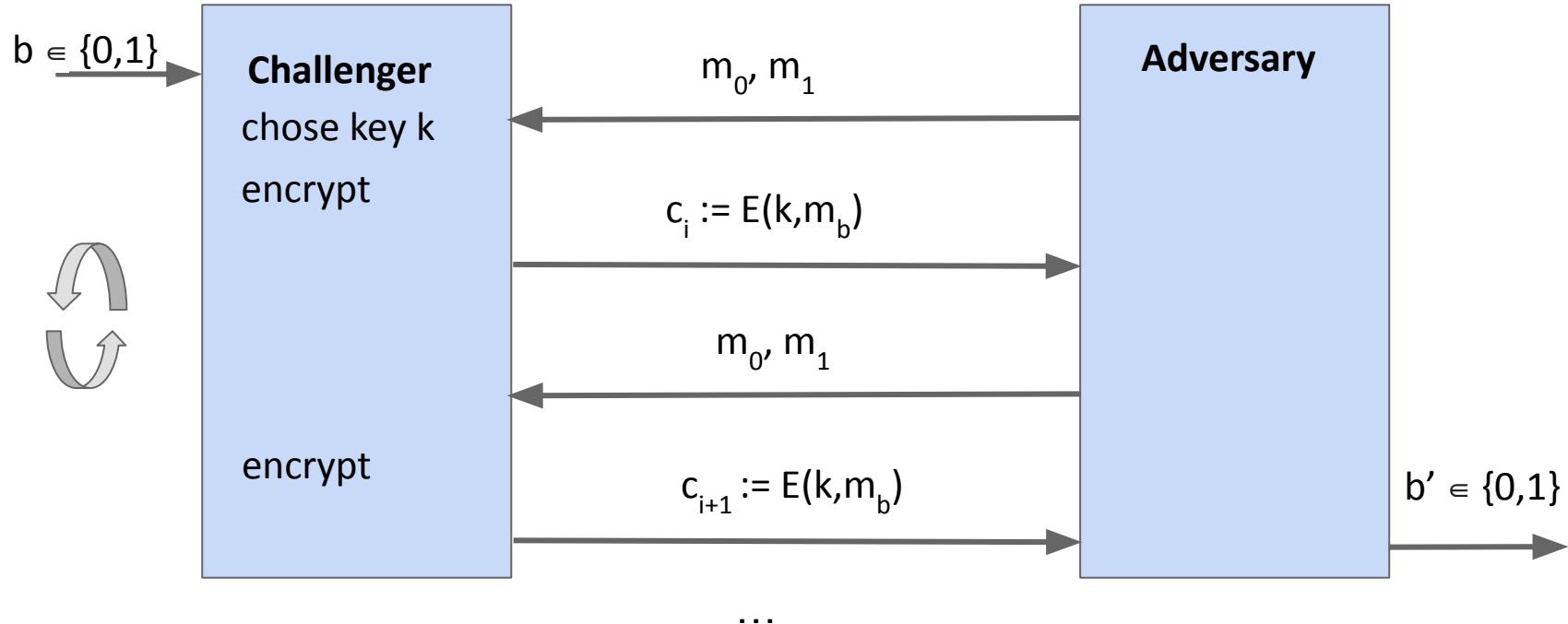
The advantage of the attacker to distinguish between being EXP 0 or EXP 1 is negligible



The Formal Approach: Example of a Game Based Proof

Semantic security under ciphertext only attacks (COA)

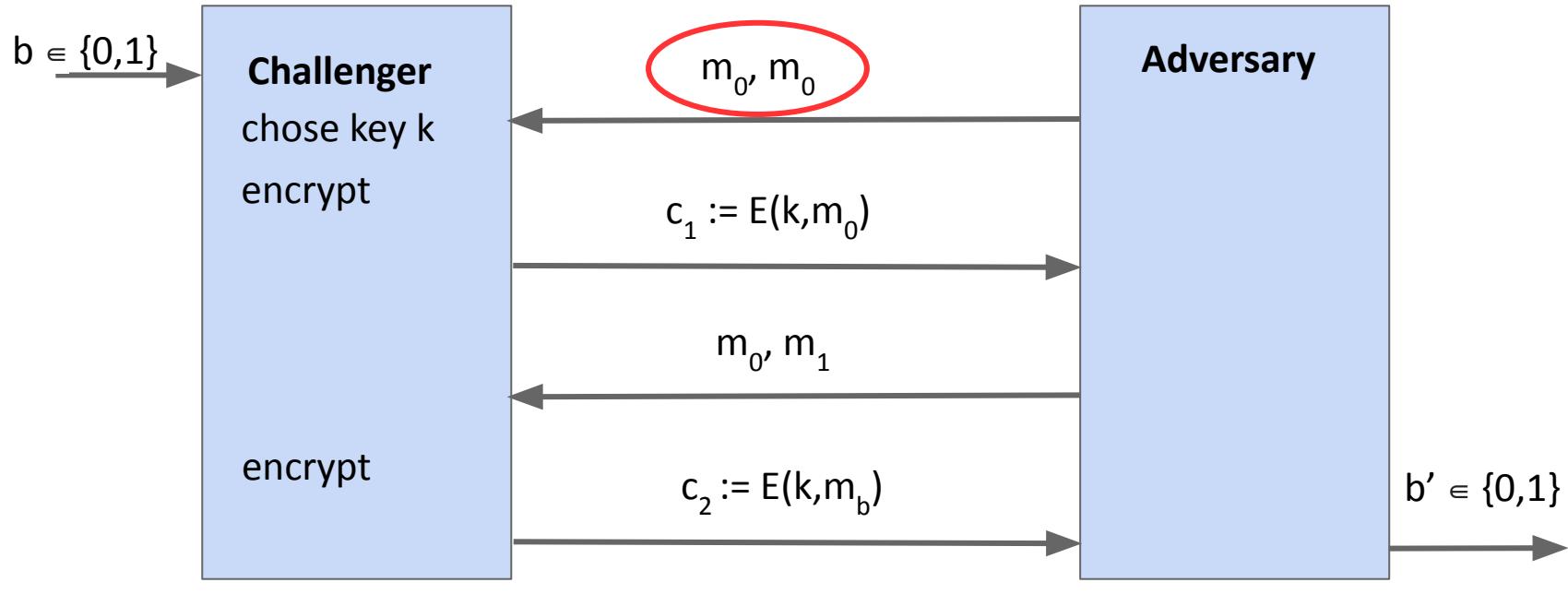
The advantage of the attacker to distinguish between being EXP 0 or EXP 1 is negligible



The Formal Approach: Example of a Game Based Proof

Semantic security does **not** hold under chosen plaintext attacks (CPA)

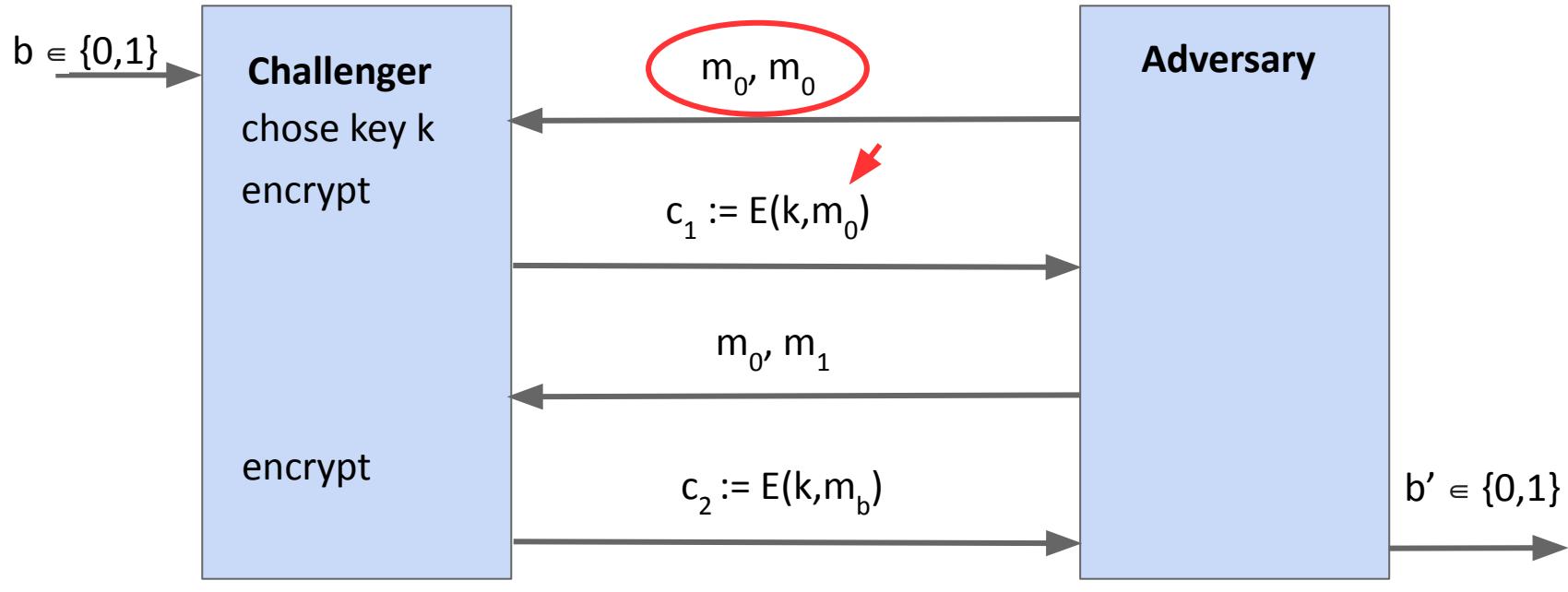
If $m_0 = m_1$ then attacker can distinguish between EXP 0 and 1



The Formal Approach: Example of a Game Based Proof

Semantic security does **not** hold under chosen plaintext attacks (CPA)

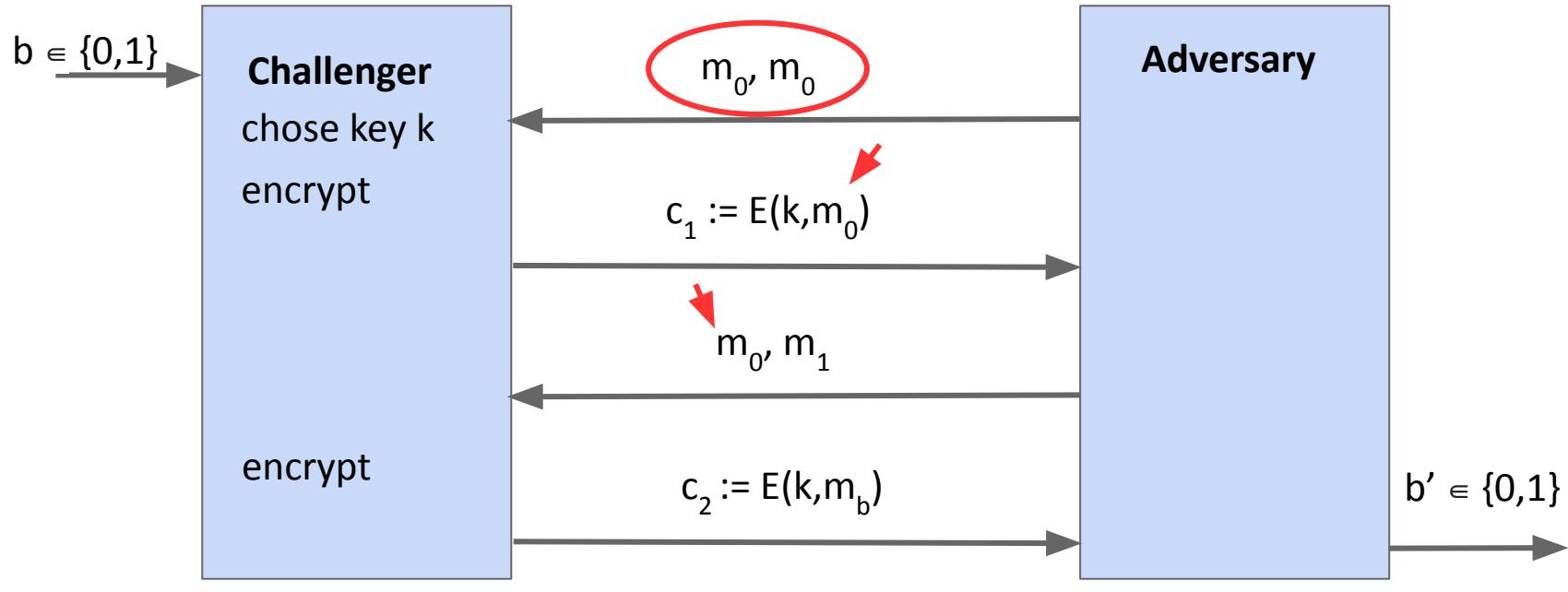
If $m_0 = m_1$ then attacker can distinguish between EXP 0 and 1



The Formal Approach: Example of a Game Based Proof

Semantic security does **not** hold under chosen plaintext attacks (CPA)

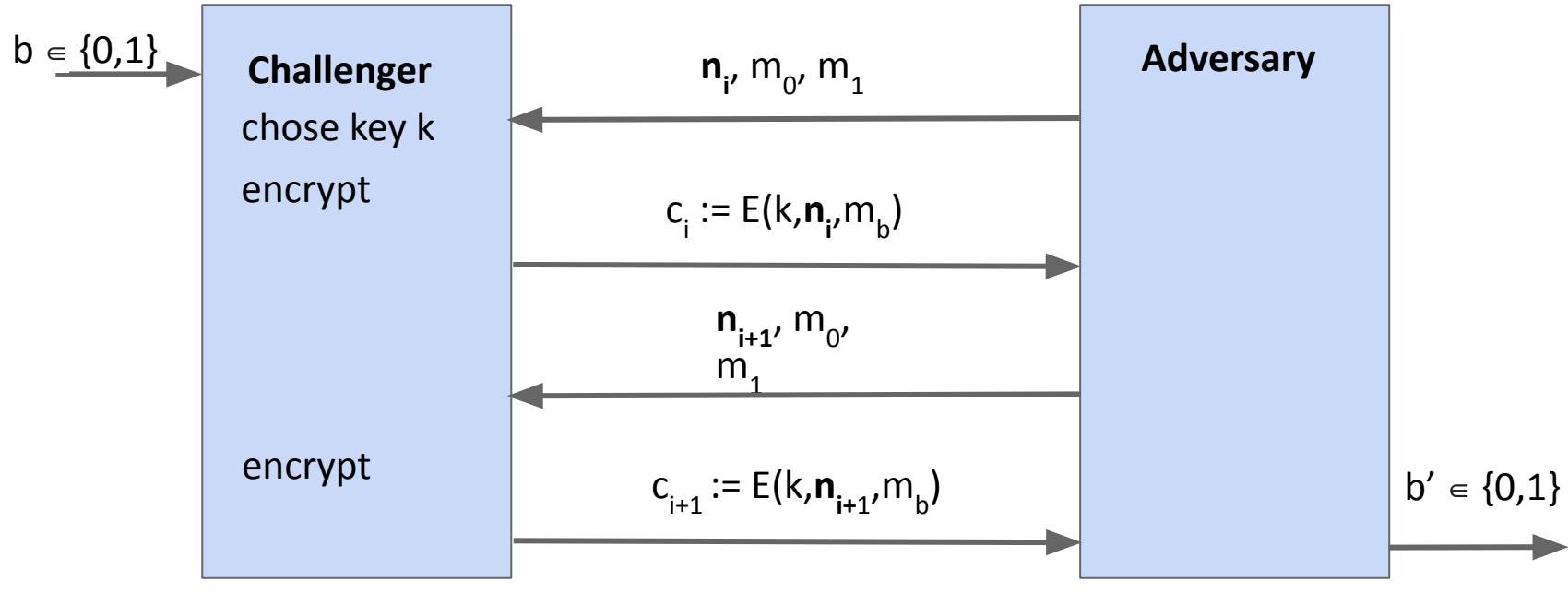
If $m_0 = m_1$ then attacker can distinguish between EXP 0 and 1



The Formal Approach: Example of a Game Based Proof

Semantic security under chosen plaintext attacks (CPA)

Theoretical fix, require a nonce which **must be distinct** and sufficiently large



The Formal Approach: Principles of modern Cryptography [1]

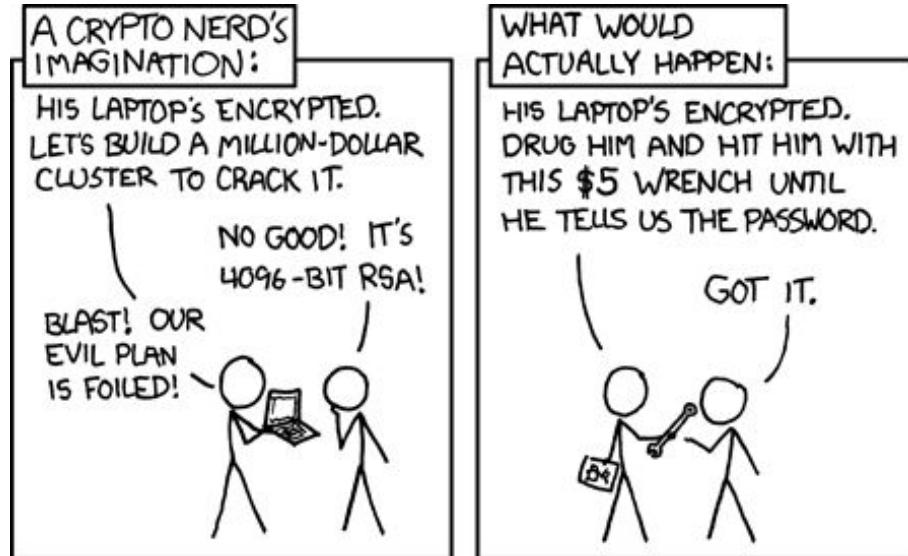
- Proofs of security under clear definitions and assumptions are important
 - Security goals/claims can be checked/verified by others
 - Changes (swapping out certain cryptographic primitives) are easier ⇒ different algorithms and implementations can be used to achieve goals
- A security proof is a good start, but it does **not necessarily** imply the security of a scheme in the real world
 - **Implementation weakness**
memory corruption vulnerability, injection vulnerability, ...
 - Scheme might be secure, but **used incorrectly** within the context of the system as a whole
SHA2 is a secure hash function but not a MAC, secret key leaked on camera, ...
 - Underlying **assumptions** turn out to be **wrong**
MD5 is secure hash function, Dual_EC_DRBG is not backdoored, ...
 - **Inappropriate definitions** which do not capture important real world requirements
Something was not explicitly considered in the threat model (burning Zerocoins of others, ...)



Theory VS Practice



Theory VS Practice



“Civil engineers learn far more from the one bridge that falls down than from the hundred that stay up; exactly the same holds in security engineering.” [1]

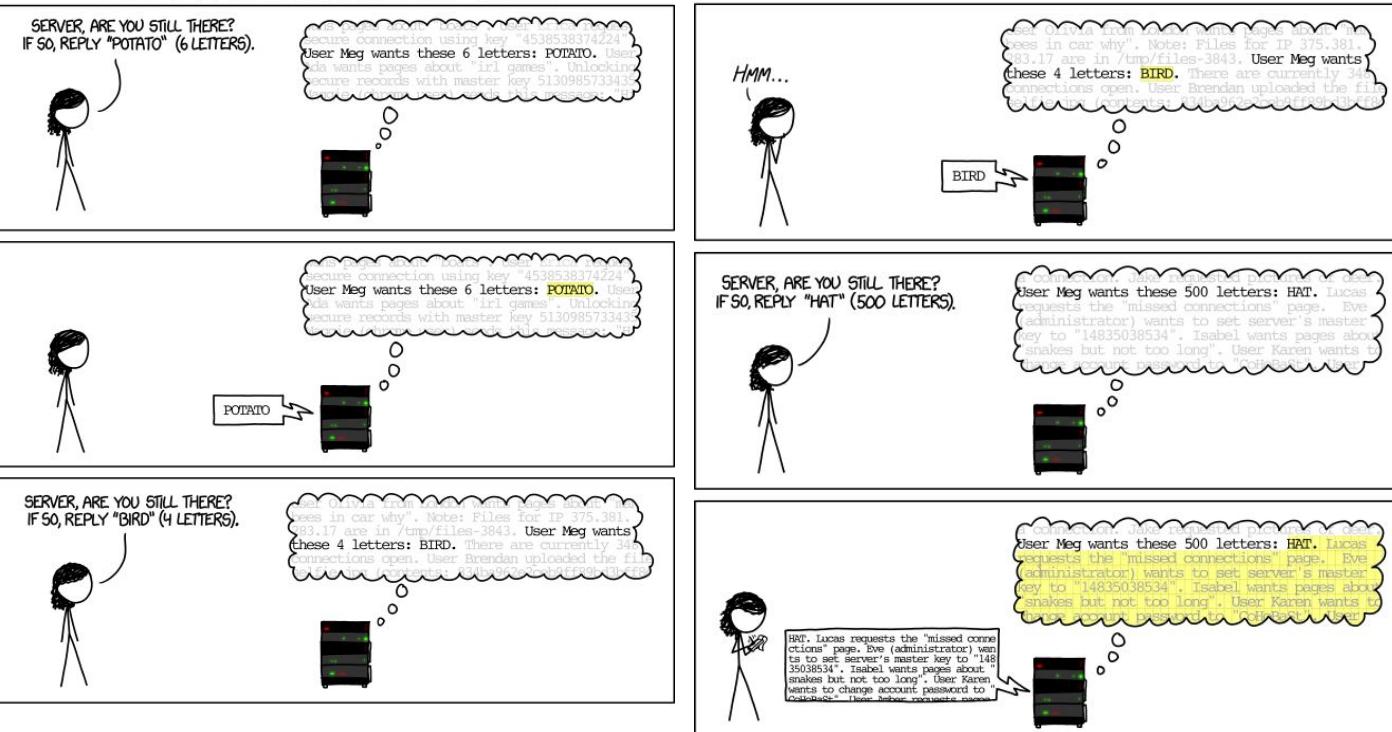
[1] Ross Anderson, Security Engineering, 3rd Edition



Heartbleed (impl. flaw)

impl.

HOW THE HEARTBLEED BUG WORKS:



def.

Side channels (EM)

- ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels



Mounting a cheap EM attack on an iPhone 4 using an improvised EM probe. The target (top right) is measured by the improvised probe (taped to the underside of a glass table). The signal is captured by a Tracker Pre sound card connected to a laptop (under the table).

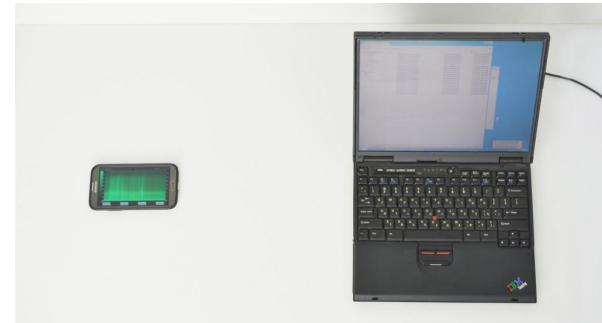


Improvised probe (view from under the glass table).

def.

Side channels (acoustic)

- “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”
 - “The acoustic signal of interest is generated by vibration of electronic components (capacitors and coils) in the computer's voltage regulation circuit, as it struggles to supply constant voltage to the CPU.”



[1] <http://www.cs.tau.ac.il/~tromer/acoustic/>

use.

Side channels (visual)

- (2013) Bloomberg anchor displays \$20 worth in Bitcoin on TV, immediately gets robbed by viewer



[1] <http://www.businessinsider.com/bloomberg-matt-miller-bitcoin-gift-stolen-2013-12?IR=T>
[2] <https://www.rt.com/usa/bloomberg-anchor-robbed-bitcoin-747/>

Crypto bypass example: Side channels (timing)

- Spectre, Meltdown, ...

Goto fail (impl. flaw)

- Apple goto fail

```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
3                                     uint8_t *signature, UInt16 signatureLen)
4 {
5     OSStatus         err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11        goto fail;
12        goto fail;
13    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14        goto fail;
15    ...
16
17 fail:
18     SSLFreeBuffer(&signedHashes);
19     SSLFreeBuffer(&hashCtx);
20     return err;
21 }
```

[1] <https://gotofail.com/>

Goto fail (impl. flaw)

- Apple goto fail

```
1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
3                                     uint8_t *signature, UInt16 signatureLen)
4 {
5     OSStatus err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11        goto fail;
12        goto fail;
13    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14        goto fail;
15    ...
16
17 fail:
18     SSLFreeBuffer(&signedHashes);
19     SSLFreeBuffer(&hashCtx);
20     return err;
21 }
```

Always reached

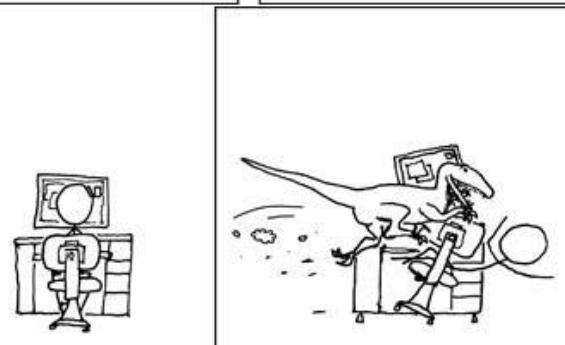
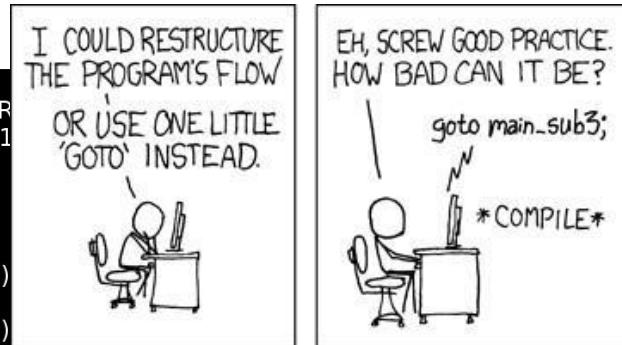
‘err’ not set therefore
all certificates are accepted

Goto fail (impl. flaw)

- Apple goto fail

```

1 static OSStatus
2 SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isR
3                                     uint8_t *signature, UInt1
4 {
5     OSStatus         err;
6     ...
7
8     if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom))
9         goto fail;
10    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams))
11        goto fail;
12        goto fail;
13    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14        goto fail;
15    ...
16
17 fail:
18     SSLFreeBuffer(&signedHashes);
19     SSLFreeBuffer(&hashCtx);
20     return err;
21 }
```



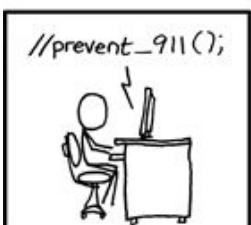
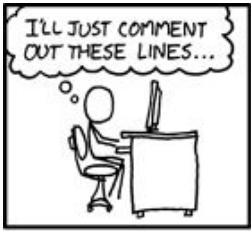
[1] <https://gotofail.com/>

Debian PRNG (impl. flaw)

- Debian PRNG bug
 - Removed source code line that was causing complains from *Valgrind*
 - `/*`
 - `* Don't add uninitialized data`
 - `MD_Update(&m,buf,j);`
 - `*/`
 - **Problem:** This uninitialized data was mixed in the seed of the PRNG. Therefore the only randomness left was the PID which is in the range 2–32768
 - “Affected keys include **SSH keys**, **OpenVPN keys**, **DNSSEC keys**, and key material for use in **X.509 certificates** and **session keys used in SSL/TLS connections**.”

[1] <https://www.debian.org/security/2008/dsa-1571>

[2] https://www.schneier.com/blog/archives/2008/05/random_number_b.html



IN THE RUSH TO CLEAN UP THE DEBIAN-OPENSSL FIASCO, A NUMBER OF OTHER MAJOR SECURITY HOLES HAVE BEEN UNCOVERED:

AFFECTED SYSTEM SECURITY PROBLEM

FEDORA CORE	VULNERABLE TO CERTAIN DECODER RINGS
XANDROS (EEE PC)	GIVES ROOT ACCESS IF ASKED IN STERN VOICE
GENTOO	VULNERABLE TO FLATTERY
OLPC OS	VULNERABLE TO JEFF GOLDBLUM'S POWERBOOK
SLACKWARE	GIVES ROOT ACCESS IF USER SAYS ELVISH WORD FOR "FRIEND"
UBUNTU	TURNS OUT DISTRO IS ACTUALLY JUST WINDOWS VISTA WITH A FEW CUSTOM THEMES

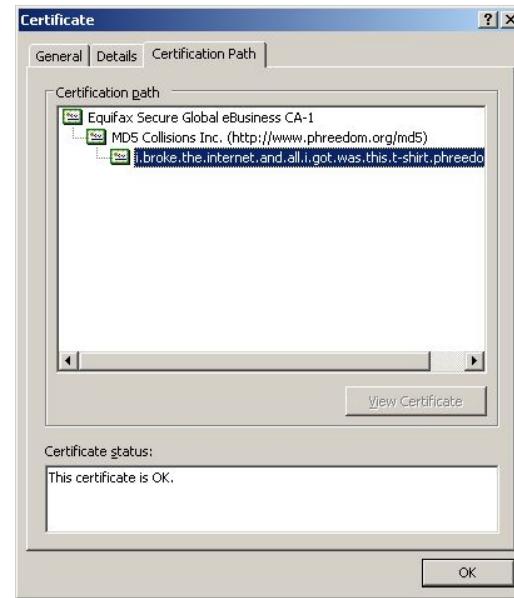
Good news: Getting cryptography right is also hard for malware authors!

- 2015: Linux.encoder ransomware
 - Used srand(time()) as random key for encryption
 - Then they used 8x srand(md5(time())) as key for encryption
- 2014: CryptoDefense ransomware
 - Did not implement encryption functionally themselves
 - Used low-level cryptographic API offered by Windows OS for RSA encryption
 - Used CryptAcquireContext in a way that persisted the created public/private key pairs safely in the users in the local key-store
- 2016: NotPetya/Petya
 - Implemented Salsa20 on their own (no common lib used)
 - Among other errors, used 32bit or 16bit values on different occasions instead of 64bit values reducing the key size from 512 to 256 bits.
- ...

assum.

MD5

- “MD5 considered harmful today: Creating a rogue CA certificate”
 - Back in **2009**
 - Created a rogue Certification Authority (CA) trusted by all common web browsers as a proof of concept certificate
 - Used 200 PS 3 to **create a MD5 collision**
 - Build on:
 - Previous work on MD5 collisions
 - Previous work on creating collision on certificate data (from 2004 and 2007) which showed theoretical attack scenarios

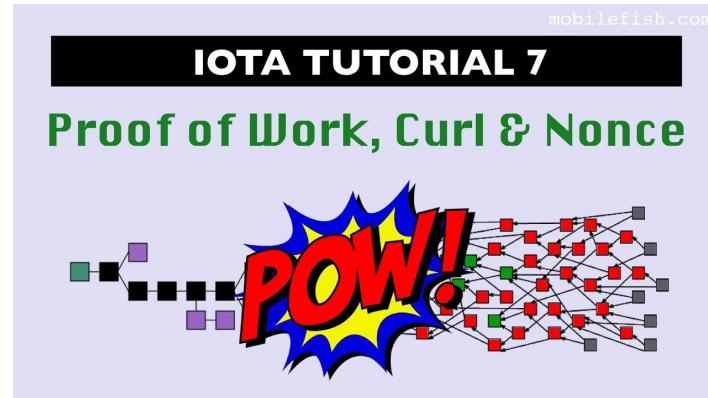


[1] <http://www.win.tue.nl/hashclash/rogue-ca/>

[2] <https://documents.epfl.ch/users/l/le/lenstra/public/papers/lat.pdf>

Curl hash function

- 2017: Practical attacks on IOTA's (custom) cryptographic hash function Curl
- “*under certain conditions the ability to forge signatures*”
- “*Curl should not be relied on for randomness or collision resistance*”



[1] <https://github.com/mit-dci/tangled-curl/blob/master/vuln-iota.md>

[2] <http://ethanheilman.tumblr.com/post/165085348035/breaking-the-crypto-used-in-iota>

[3] <https://rwc.iacr.org/2019/slides/neha.pdf>

Curl hash function

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES S-Box

-1	1	1	-1
0	0	-1	1
1	-1	0	0

Curl-P-27 S-Box

SwissPost e-voting

- “*The implementation of the commitment scheme in the SwissPost-Scytl mixnet uses a **trapdoor commitment scheme**, which allows anyone who knows the trapdoor values to generate a shuffle proof transcript that passes verification but actually alters votes. **This allows undetectable vote manipulation by an authority who implemented or administered a mix server.***”
- We give two examples of how an authority who implemented or administered a mix server could produce a perfectly-verifying transcript while actually - undetectably - manipulating votes.

[1] <https://people.eng.unimelb.edu.au/vjteague/SwissVote>

[2] <https://e-voting.bfh.ch/publications/2019/>

def.

Burning Zerocoins

- “In this article, we present a cryptographic flaw in the Zerocoins cryptographic scheme (not Zerocash), which allows an **attacker** to **burn coins of honest users**. On the way, we identified **two more critical coding issues** in a software library implementing Zerocoins, allowing an attacker to **create money out of thin air and stealing coins** from honest users.”

Burning Zerocoins for Fun and for Profit

A Cryptographic Denial-of-Spending Attack on the Zerocoins Protocol

Tim Ruffing

Saarland University, Germany
tim.ruffing@mmtc.uni-saarland.de

Sri Aravinda Thyagarajan

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
name.separated.by.dots@fau.de

Viktoria Ronge

Dominique Schröder



universität
wien

SBA
Research

[1] <https://www.chaac.tf.fau.eu/2018/04/12/zerocoinscoinpivxzoinsmartcashhexxcoin-attack/>

[2] <https://www.chaac.tf.fau.de/files/2018/04/attack-cryptocur.pdf>

Zcash

- “*The vulnerability was specific to counterfeiting and did not affect user privacy in any way. Prior to its remediation, an attacker could have **created fake Zcash without being detected**. The counterfeiting vulnerability has been fully remediated in Zcash and no action is required by Zcash users.*”
- On March 1, 2018, Ariel Gabizon, a cryptographer employed by the Zcash Company at the time, discovered a subtle cryptographic flaw in the [2] paper that describes the zk-SNARK construction used in the original launch of Zcash. The flaw allows an attacker to create counterfeit shielded value in any system that depends on parameters which are generated as described by the paper.

[1] <https://z.cash/blog/zcash-counterfeiting-vulnerability-successfully-remediated/>

[2] <https://eprint.iacr.org/2013/879>

Zcash



Public parameters. A prime r , two cyclic groups \mathbb{G}_1 and \mathbb{G}_2 of order r with generators \mathcal{P}_1 and \mathcal{P}_2 respectively, and a pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ (where \mathbb{G}_T is also cyclic of order r).

(a) Key generator G

- INPUTS: circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$
- OUTPUTS: proving key pk and verification key vk

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$; extend $\vec{A}, \vec{B}, \vec{C}$ via

$$A_{m+1} = B_{m+2} = C_{m+3} = Z,$$

$$A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0.$$

2. Randomly sample $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma \in \mathbb{F}_r$.

3. Set $\text{pk} := (C, \text{pk}_A, \text{pk}'_A, \text{pk}_B, \text{pk}'_B, \text{pk}_C, \text{pk}'_C, \text{pk}_H, \text{pk}_H)$ where for $i = 0, 1, \dots, m+3$:

$$\begin{aligned} \text{pk}_{A,i} &:= A_i(\tau)\rho_A\mathcal{P}_1, \quad \text{pk}'_{A,i} := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1, \\ \text{pk}_{B,i} &:= B_i(\tau)\rho_B\mathcal{P}_2, \quad \text{pk}'_{B,i} := B_i(\tau)\alpha_B\rho_B\mathcal{P}_2, \\ \text{pk}_{C,i} &:= C_i(\tau)\rho_A\rho_B\mathcal{P}_1, \quad \text{pk}'_{C,i} := C_i(\tau)\alpha_C\rho_A\rho_B\mathcal{P}_1, \\ \text{pk}_{H,i} &:= \beta(A_i(\tau)\rho_A + B_i(\tau)\rho_B + C_i(\tau)\rho_A\rho_B)\mathcal{P}_1, \end{aligned}$$

and for $i = 0, 1, \dots, d$, $\text{pk}_{H,i} := \tau^i\mathcal{P}_1$.

4. Set $\text{vk} := (\text{vk}_A, \text{vk}_B, \text{vk}_C, \text{vk}_\gamma, \text{vk}_{\beta\gamma}, \text{vk}_{\beta^2\gamma}, \text{vk}_Z, \text{vk}_C)$ where

$$\text{vk}_A := \alpha_B\mathcal{P}_2, \quad \text{vk}_B := \alpha_B\mathcal{P}_1, \quad \text{vk}_C := \alpha_C\mathcal{P}_2$$

$$\text{vk}_\gamma := \gamma\mathcal{P}_2, \quad \text{vk}_{\beta\gamma}^1 := \gamma\beta\mathcal{P}_1, \quad \text{vk}_{\beta^2\gamma}^2 := \gamma\beta\mathcal{P}_2,$$

$$\text{vk}_Z := Z(\tau)\rho_A\rho_B\mathcal{P}_2, \quad (\text{vk}_{C,i})_{i=0}^n := (A_i(\tau)\rho_A\mathcal{P}_1)_{i=0}^n.$$

5. Output (pk, vk) .

Key sizes. When invoked on a circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$ with a wires and b (bilinear) gates, the key generator outputs:

- pk with $(6a + b + n + l + 26)$ \mathbb{G}_1 -elements and $(a + 4)$ \mathbb{G}_2 -elements;
- vk with $(n + 3)$ \mathbb{G}_1 -elements and 5 \mathbb{G}_2 -elements.

Proof size. The proof always has 7 \mathbb{G}_1 -elements and 1 \mathbb{G}_2 -element.

[1] <https://eprint.iacr.org/archive/2013/879/1432056364.pdf> (bugged)

[2] <https://eprint.iacr.org/archive/2013/879/1549383918.pdf> (fixed)

(b) Prover P

- INPUTS: proving key pk , input $\vec{x} \in \mathbb{F}_r^n$, and witness $\vec{a} \in \mathbb{F}_r^h$
- OUTPUTS: proof π

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$.

2. Compute $\vec{s} := \text{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$.

3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$.

4. Compute $\vec{h} = (h_0, h_1, \dots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)-C(z)}{Z(z)}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:

$$A(z) := A_0(z) + \sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z),$$

$$B(z) := B_0(z) + \sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z),$$

$$C(z) := C_0(z) + \sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z).$$

5. Set $\text{pk}_A := \text{"same as } \text{pk}_A\text{, but with } \text{pk}_{A,i} = 0 \text{ for } i = 0, 1, \dots, n\text{"}$.

Set $\text{pk}'_A := \text{"same as } \text{pk}'_A\text{, but with } \text{pk}'_{A,i} = 0 \text{ for } i = 0, 1, \dots, n\text{"}$.

6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, compute

$$\pi_A := \langle \vec{c}, \text{pk}_A \rangle, \quad \pi'_A := \langle \vec{c}, \text{pk}'_A \rangle, \quad \pi_B := \langle \vec{c}, \text{pk}_B \rangle, \quad \pi'_B := \langle \vec{c}, \text{pk}'_B \rangle,$$

$$\pi_C := \langle \vec{c}, \text{pk}_C \rangle, \quad \pi'_C := \langle \vec{c}, \text{pk}'_C \rangle, \quad \pi_K := \langle \vec{c}, \text{pk}_H \rangle, \quad \pi_H := \langle \vec{c}, \text{pk}_H \rangle.$$

7. Output $\pi := (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H)$.

(c) Verifier V

- INPUTS: verification key vk , input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\text{vk}_Z := \text{vk}_{Z,0} + \sum_{i=1}^n x_i \text{vk}_{C,i} \in \mathbb{G}_1$.

2. Check validity of knowledge commitments for A, B, C :

$$e(\pi_A, \text{vk}_A) = e(\pi'_A, \mathcal{P}_2), \quad e(\text{vk}_B, \pi_B) = e(\pi_B, \mathcal{P}_2),$$

$$e(\pi_C, \text{vk}_C) = e(\pi'_C, \mathcal{P}_2).$$

3. Check same coefficients were used:

$$e(\pi_K, \text{vk}_\gamma) = e(\text{vk}_Z + \pi_A + \pi_C, \text{vk}_{\beta\gamma}^1) \cdot e(\text{vk}_{\beta\gamma}^1, \pi_B).$$

4. Check QAP divisibility:

$$e(\text{vk}_Z + \pi_A, \pi_B) = e(\text{vk}_Z, \pi_B) \cdot e(\pi_A, \mathcal{P}_2).$$

5. Accept if and only if all the above checks succeeded.

Public parameters. A prime r , two cyclic groups \mathbb{G}_1 and \mathbb{G}_2 of order r with generators \mathcal{P}_1 and \mathcal{P}_2 respectively, and a pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ (where \mathbb{G}_T is also cyclic of order r).

(a) Key generator G

- INPUTS: circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$
- OUTPUTS: proving key pk and verification key vk

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$; extend $\vec{A}, \vec{B}, \vec{C}$ via

$$A_{m+1} = B_{m+2} = C_{m+3} = Z,$$

$$A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0.$$

2. Randomly sample $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma \in \mathbb{F}_r$.

3. Set $\text{pk} := (C, \text{pk}_A, \text{pk}'_A, \text{pk}_B, \text{pk}'_B, \text{pk}_C, \text{pk}'_C, \text{pk}_H, \text{pk}_H)$ where:

$$\text{pk}_A := \{A_i(\tau)\rho_A\mathcal{P}_1\}_{i=0}^{m+3}, \quad \text{pk}'_A := \{A_i(\tau)\alpha_A\rho_A\mathcal{P}_1\}_{i=0}^{m+3}$$

$$\text{pk}_B := \{B_i(\tau)\rho_B\mathcal{P}_2\}_{i=0}^{m+3}, \quad \text{pk}'_B := \{B_i(\tau)\alpha_B\rho_B\mathcal{P}_2\}_{i=0}^{m+3},$$

$$\text{pk}_C := \{C_i(\tau)\rho_A\rho_B\mathcal{P}_1\}_{i=0}^{m+3}, \quad \text{pk}'_C := \{C_i(\tau)\alpha_C\rho_A\rho_B\mathcal{P}_1\}_{i=0}^{m+3},$$

$$\text{pk}_H := \{\beta(A_i(\tau)\rho_A + B_i(\tau)\rho_B + C_i(\tau)\rho_A\rho_B)\mathcal{P}_1\}_{i=0}^{m+3},$$

and $\text{pk}_H := \{\tau^i\mathcal{P}_1\}_{i=0}^d$.

4. Set $\text{vk} := (\text{vk}_A, \text{vk}_B, \text{vk}_C, \text{vk}_\gamma, \text{vk}_{\beta\gamma}, \text{vk}_Z, \text{vk}_C)$ where

$$\text{vk}_A := \alpha_B\mathcal{P}_2, \quad \text{vk}_B := \alpha_B\mathcal{P}_1, \quad \text{vk}_C := \alpha_C\mathcal{P}_2$$

$$\text{vk}_\gamma := \gamma\mathcal{P}_2, \quad \text{vk}_{\beta\gamma}^1 := \gamma\beta\mathcal{P}_1, \quad \text{vk}_{\beta^2\gamma}^2 := \gamma\beta\mathcal{P}_2,$$

$$\text{vk}_Z := Z(\tau)\rho_A\rho_B\mathcal{P}_2, \quad \text{vk}_C := (A_i(\tau)\rho_A\mathcal{P}_1)_{i=0}^n.$$

5. Output (pk, vk) .

Key sizes. When invoked on a circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$ with a wires and b (bilinear) gates, the key generator outputs:

- pk with $(6a + b + l + 25)$ \mathbb{G}_1 -elements and $(a + 4)$ \mathbb{G}_2 -elements;
- vk with $(n + 3)$ \mathbb{G}_1 -elements and 5 \mathbb{G}_2 -elements.

Proof size. The proof always has 7 \mathbb{G}_1 -elements and 1 \mathbb{G}_2 -element.

(b) Prover P

- INPUTS: proving key pk , input $\vec{x} \in \mathbb{F}_r^n$, and witness $\vec{a} \in \mathbb{F}_r^h$
- OUTPUTS: proof π

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$.

2. Compute $\vec{s} := \text{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$.

3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$.

4. Compute $\vec{h} = (h_0, h_1, \dots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)-C(z)}{Z(z)}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:

$$A(z) := A_0(z) + \sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z),$$

$$B(z) := B_0(z) + \sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z),$$

$$C(z) := C_0(z) + \sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z).$$

5. Set $\text{pk}_A := \text{"same as } \text{pk}_A\text{, but with } \text{pk}_{A,i} = 0 \text{ for } i = 0, 1, \dots, n\text{"}$.

Set $\vec{\text{pk}}_A := \text{"same as } \text{pk}_A\text{, but prepend } n+1 \text{ zeroes"}$.

6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, compute

$$\pi_A := \langle \vec{c}, \text{pk}_A \rangle, \quad \pi'_A := \langle \vec{c}, \text{pk}'_A \rangle, \quad \pi_B := \langle \vec{c}, \text{pk}_B \rangle, \quad \pi'_B := \langle \vec{c}, \text{pk}'_B \rangle,$$

$$\pi_C := \langle \vec{c}, \text{pk}_C \rangle, \quad \pi'_C := \langle \vec{c}, \text{pk}'_C \rangle, \quad \pi_K := \langle \vec{c}, \text{pk}_H \rangle, \quad \pi_H := \langle \vec{c}, \text{pk}_H \rangle.$$

7. Output $\pi := (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H)$.

(c) Verifier V

- INPUTS: verification key vk , input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\text{vk}_Z := \text{vk}_{Z,0} + \sum_{i=1}^n x_i \text{vk}_{C,i} \in \mathbb{G}_1$.

2. Check validity of knowledge commitments for A, B, C :

$$e(\pi_A, \text{vk}_A) = e(\pi'_A, \mathcal{P}_2), \quad e(\text{vk}_B, \pi_B) = e(\pi_B, \mathcal{P}_2),$$

$$e(\pi_C, \text{vk}_C) = e(\pi'_C, \mathcal{P}_2).$$

3. Check same coefficients were used:

$$e(\pi_K, \text{vk}_\gamma) = e(\text{vk}_Z + \pi_A + \pi_C, \text{vk}_{\beta\gamma}^1) \cdot e(\text{vk}_{\beta\gamma}^1, \pi_B).$$

4. Check QAP divisibility:

$$e(\text{vk}_Z + \pi_A, \pi_B) = e(\text{vk}_Z, \pi_B) \cdot e(\pi_A, \mathcal{P}_2).$$

5. Accept if and only if all the above checks succeeded.



universität
wien

SBA
Research

Zcash



Public parameters. A prime r , two cyclic groups \mathbb{G}_1 and \mathbb{G}_2 of order r with generators \mathcal{P}_1 and \mathcal{P}_2 respectively, and a pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ (where \mathbb{G}_T is also cyclic of order r).

(a) Key generator G

- INPUTS: circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$
- OUTPUTS: proving key pk and verification key vk

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$; extend $\vec{A}, \vec{B}, \vec{C}$ via

$$A_{m+1} = B_{m+2} = C_{m+3} = Z,$$

$$A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0.$$

2. Randomly sample $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma \in \mathbb{F}_r$.

3. Set $\text{pk} := (C, \text{pk}_A, \text{pk}'_A, \text{pk}_B, \text{pk}'_B, \text{pk}_C, \text{pk}'_C, \text{pk}_H, \text{pk}_I)$ where for $i = 0, 1, \dots, m+3$:

$$\begin{aligned} \text{pk}_{A,i} &:= A_i(\tau)\rho_A\mathcal{P}_1, \quad \text{pk}'_{A,i} := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1, \\ \text{pk}_{B,i} &:= B_i(\tau)\rho_B\mathcal{P}_2, \quad \text{pk}'_{B,i} := B_i(\tau)\alpha_B\rho_B\mathcal{P}_2, \\ \text{pk}_{C,i} &:= C_i(\tau)\rho_A\rho_B\mathcal{P}_1, \quad \text{pk}'_{C,i} := C_i(\tau)\alpha_C\rho_A\rho_B\mathcal{P}_1, \\ \text{pk}_{H,i} &:= \beta(A_i(\tau)\rho_A + B_i(\tau)\rho_B + C_i(\tau)\rho_A\rho_B)\mathcal{P}_1, \end{aligned}$$

and for $i = 0, 1, \dots, d$, $\text{pk}_{H,i} := \tau^i\mathcal{P}_1$.

4. Set $\text{vk} := (\text{vk}_A, \text{vk}_B, \text{vk}_C, \text{vk}_\gamma, \text{vk}_{\beta\gamma}, \text{vk}_{\beta^2\gamma}, \text{vk}_Z, \text{vk}_C)$ where

$$\text{vk}_A := \alpha_A\mathcal{P}_2, \quad \text{vk}_B := \alpha_B\mathcal{P}_1, \quad \text{vk}_C := \alpha_C\mathcal{P}_2$$

$$\text{vk}_\gamma := \gamma\mathcal{P}_2, \quad \text{vk}_{\beta\gamma}^1 := \gamma\beta\mathcal{P}_1, \quad \text{vk}_{\beta^2\gamma}^2 := \gamma\beta\mathcal{P}_2,$$

$$\text{vk}_Z := Z(\tau)\rho_A\rho_B\mathcal{P}_2, \quad (\text{vk}_{C,i})_{i=0}^n := (A_i(\tau)\rho_A\mathcal{P}_1)_{i=0}^n.$$

5. Output (pk, vk) .

Key sizes. When invoked on a circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$ with a wires and b (bilinear) gates, the key generator outputs:

- pk with $(6a + b + n + l + 26)$ \mathbb{G}_1 -elements and $(a + 4)$ \mathbb{G}_2 -elements;
- vk with $(n + 3)$ \mathbb{G}_1 -elements and 5 \mathbb{G}_2 -elements.

Proof size. The proof always has 7 \mathbb{G}_1 -elements and 1 \mathbb{G}_2 -element.

[1] <https://eprint.iacr.org/archive/2013/879/1432056364.pdf> (bugged)

[2] <https://eprint.iacr.org/archive/2013/879/1549383918.pdf> (fixed)

(b) Prover P

- INPUTS: proving key pk , input $\vec{x} \in \mathbb{F}_r^n$, and witness $\vec{a} \in \mathbb{F}_r^h$
- OUTPUTS: proof π

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$.

2. Compute $\vec{s} := \text{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$.

3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$.

4. Compute $\vec{h} = (h_0, h_1, \dots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)-C(z)}{Z(z)}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:

$$A(z) := A_0(z) + \sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z),$$

$$B(z) := B_0(z) + \sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z),$$

$$C(z) := C_0(z) + \sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z).$$

5. Set $\text{pk}_A := \text{"same as } \text{pk}_A\text{, but with } \text{pk}_{A,i} = 0 \text{ for } i = 0, 1, \dots, n\text{"}$.

Set $\text{pk}'_A := \text{"same as } \text{pk}'_A\text{, but with } \text{pk}'_{A,i} = 0 \text{ for } i = 0, 1, \dots, n\text{"}$.

6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, compute

$$\pi_A := \langle \vec{c}, \text{pk}_A \rangle, \quad \pi'_A := \langle \vec{c}, \text{pk}'_A \rangle, \quad \pi_B := \langle \vec{c}, \text{pk}_B \rangle, \quad \pi'_B := \langle \vec{c}, \text{pk}'_B \rangle,$$

$$\pi_C := \langle \vec{c}, \text{pk}_C \rangle, \quad \pi'_C := \langle \vec{c}, \text{pk}'_C \rangle, \quad \pi_K := \langle \vec{c}, \text{pk}_K \rangle, \quad \pi_H := \langle \vec{c}, \text{pk}_H \rangle.$$

7. Output $\pi := (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H)$.

(c) Verifier V

- INPUTS: verification key vk , input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\text{vk}_Z := \text{vk}_{Z,0} + \sum_{i=1}^n x_i \text{vk}_{C,i} \in \mathbb{G}_1$.

2. Check validity of knowledge commitments for A, B, C :

$$e(\pi_A, \text{vk}_A) = e(\pi'_A, \mathcal{P}_2), \quad e(\text{vk}_B, \pi_B) = e(\pi_B, \mathcal{P}_2),$$

$$e(\pi_C, \text{vk}_C) = e(\pi'_C, \mathcal{P}_2).$$

3. Check same coefficients were used:

$$e(\pi_K, \text{vk}_\gamma) = e(\text{vk}_Z + \pi_A + \pi_C, \text{vk}_{\beta\gamma}^1) \cdot e(\text{vk}_{\beta\gamma}^1, \pi_B).$$

4. Check QAP divisibility:

$$e(\text{vk}_Z + \pi_A, \pi_B) = e(\text{vk}_Z, \pi_B) \cdot e(\pi_A, \mathcal{P}_2).$$

5. Accept if and only if all the above checks succeeded.

Public parameters. A prime r , two cyclic groups \mathbb{G}_1 and \mathbb{G}_2 of order r with generators \mathcal{P}_1 and \mathcal{P}_2 respectively, and a pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ (where \mathbb{G}_T is also cyclic of order r).

(a) Key generator G

- INPUTS: circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$
- OUTPUTS: proving key pk and verification key vk

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$; extend $\vec{A}, \vec{B}, \vec{C}$ via

$$A_{m+1} = B_{m+2} = C_{m+3} = Z,$$

$$A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0.$$

2. Randomly sample $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma \in \mathbb{F}_r$.

3. Set $\text{pk} := (C, \text{pk}_A, \text{pk}'_A, \text{pk}_B, \text{pk}'_B, \text{pk}_C, \text{pk}'_C, \text{pk}_H)$ where:

$$\text{pk}_A := \{A_i(\tau)\rho_A\mathcal{P}_1\}_{i=0}^{m+3}, \quad \text{pk}'_A := \{A_i(\tau)\alpha_A\rho_A\mathcal{P}_1\}_{i=0}^{m+3}$$

$$\text{pk}_B := \{B_i(\tau)\rho_B\mathcal{P}_2\}_{i=0}^{m+3}, \quad \text{pk}'_B := \{B_i(\tau)\alpha_B\rho_B\mathcal{P}_2\}_{i=0}^{m+3},$$

$$\text{pk}_C := \{C_i(\tau)\rho_A\rho_B\mathcal{P}_1\}_{i=0}^{m+3}, \quad \text{pk}'_C := \{C_i(\tau)\alpha_C\rho_A\rho_B\mathcal{P}_1\}_{i=0}^{m+3},$$

$$\text{pk}_H := \{\beta(A_i(\tau)\rho_A + B_i(\tau)\rho_B + C_i(\tau)\rho_A\rho_B)\mathcal{P}_1\}_{i=0}^{m+3},$$

and $\text{pk}_H := \{\tau^i\mathcal{P}_1\}_{i=0}^d$.

4. Set $\text{vk} := (\text{vk}_A, \text{vk}_B, \text{vk}_C, \text{vk}_\gamma, \text{vk}_{\beta\gamma}, \text{vk}_Z, \text{vk}_C)$ where

$$\text{vk}_A := \alpha_A\mathcal{P}_2, \quad \text{vk}_B := \alpha_B\mathcal{P}_1, \quad \text{vk}_C := \alpha_C\mathcal{P}_2$$

$$\text{vk}_\gamma := \gamma\mathcal{P}_2, \quad \text{vk}_{\beta\gamma}^1 := \gamma\beta\mathcal{P}_1, \quad \text{vk}_{\beta^2\gamma}^2 := \gamma\beta\mathcal{P}_2,$$

$$\text{vk}_Z := Z(\tau)\rho_A\rho_B\mathcal{P}_2, \quad \text{vk}_C := (A_i(\tau)\rho_A\mathcal{P}_1)_{i=0}^n.$$

5. Output (pk, vk) .

Key sizes. When invoked on a circuit $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$ with a wires and b (bilinear) gates, the key generator outputs:

- pk with $(6a + b + l + 25)$ \mathbb{G}_1 -elements and $(a + 4)$ \mathbb{G}_2 -elements;
- vk with $(n + 3)$ \mathbb{G}_1 -elements and 5 \mathbb{G}_2 -elements.

Proof size. The proof always has 7 \mathbb{G}_1 -elements and 1 \mathbb{G}_2 -element.

(b) Prover P

- INPUTS: proving key pk , input $\vec{x} \in \mathbb{F}_r^n$, and witness $\vec{a} \in \mathbb{F}_r^h$
- OUTPUTS: proof π

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$.

2. Compute $\vec{s} := \text{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$.

3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$.

4. Compute $\vec{h} = (h_0, h_1, \dots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)-C(z)}{Z(z)}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:

$$A(z) := A_0(z) + \sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z),$$

$$B(z) := B_0(z) + \sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z),$$

$$C(z) := C_0(z) + \sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z).$$

5. Set $\text{pk}_A := \text{"same as } \text{pk}_A\text{, but with } \text{pk}_{A,i} = 0 \text{ for } i = 0, 1, \dots, n\text{"}$.

Set $\vec{\text{pk}}_A := \text{"same as } \text{pk}_A\text{, but prepend } n+1 \text{ zeroes"}$.

6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, compute

$$\pi_A := \langle \vec{c}, \text{pk}_A \rangle, \quad \pi'_A := \langle \vec{c}, \text{pk}'_A \rangle, \quad \pi_B := \langle \vec{c}, \text{pk}_B \rangle, \quad \pi'_B := \langle \vec{c}, \text{pk}'_B \rangle,$$

$$\pi_C := \langle \vec{c}, \text{pk}_C \rangle, \quad \pi'_C := \langle \vec{c}, \text{pk}'_C \rangle, \quad \pi_K := \langle \vec{c}, \text{pk}_K \rangle, \quad \pi_H := \langle \vec{c}, \text{pk}_H \rangle.$$

7. Output $\pi := (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H)$.

(c) Verifier V

- INPUTS: verification key vk , input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\text{vk}_Z := \text{vk}_{Z,0} + \sum_{i=1}^n x_i \text{vk}_{C,i} \in \mathbb{G}_1$.

2. Check validity of knowledge commitments for A, B, C :

$$e(\pi_A, \text{vk}_A) = e(\pi'_A, \mathcal{P}_2), \quad e(\text{vk}_B, \pi_B) = e(\pi_B, \mathcal{P}_2),$$

$$e(\pi_C, \text{vk}_C) = e(\pi'_C, \mathcal{P}_2).$$

3. Check same coefficients were used:

$$e(\pi_K, \text{vk}_\gamma) = e(\text{vk}_Z + \pi_A + \pi_C, \text{vk}_{\beta\gamma}^1) \cdot e(\text{vk}_{\beta\gamma}^1, \pi_B).$$

4. Check QAP divisibility:

$$e(\text{vk}_Z + \pi_A, \pi_B) = e(\text{vk}_Z, \pi_B) \cdot e(\pi_A, \mathcal{P}_2).$$

5. Accept if and only if all the above checks succeeded.



universität
wien

SBA
Research

Zcash

design

3. Set $\mathbf{pk} := (C, \mathbf{pk}_A, \mathbf{pk}'_A, \mathbf{pk}_B, \mathbf{pk}'_B, \mathbf{pk}_C, \mathbf{pk}'_C, \mathbf{pk}_K, \mathbf{pk}_H)$ where
for $i = 0, 1, \dots, m + 3$:

$$\mathbf{pk}_{A,i} := A_i(\tau) \rho_A \mathcal{P}_1, \quad \mathbf{pk}'_{A,i} := A_i(\tau) \alpha_A \rho_A \mathcal{P}_1,$$

$$\mathbf{pk}_{B,i} := B_i(\tau) \rho_B \mathcal{P}_2, \quad \mathbf{pk}'_{B,i} := B_i(\tau) \alpha_B \rho_B \mathcal{P}_1,$$

$$\mathbf{pk}_{C,i} := C_i(\tau) \rho_A \rho_B \mathcal{P}_1, \quad \mathbf{pk}'_{C,i} := C_i(\tau) \alpha_C \rho_A \rho_B \mathcal{P}_1,$$

$$\mathbf{pk}_{K,i} := \beta (A_i(\tau) \rho_A + B_i(\tau) \rho_B + C_i(\tau) \rho_A \rho_B) \mathcal{P}_1,$$

and for $i = 0, 1, \dots, d$, $\mathbf{pk}_{H,i} := \tau^i \mathcal{P}_1$.

Index should
start with 1 not
0 for \mathbf{pk}'_A

[1] <https://eprint.iacr.org/archive/2013/879/1432056364.pdf> (bugged)

[2] <https://eprint.iacr.org/archive/2013/879/1549383918.pdf> (fixed)

Theory VS Practice

- **Shamir's Law:** Crypto is bypassed, not penetrated

*“Cryptography is **usually bypassed**. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis [...] **usually there are much simpler ways of penetrating the security system**”*

Adi Shamir (The ‘S’ in RSA)



Some examples for bypass

- Side channels in general, and all the implementation flaws by malware authors
- 2008: Debian PRNG (impl. bug)
- 2011: Sony PS3 hack (ECDSA nonce reuse)
- 2013: Ubuntu 12.04 full disk encryption mode (misuse, will see later)
- 2014: Heartbleed (impl. bug)
- 2014: Goto fail (impl. bug)
- Some SSL/TLS fails
 - 2011: DigiNotar Hack & BEAST (Browser Exploit Against SSL/TLS)
 - 2012: CRIME (Compression Ratio Info-leak Made Easy)
 - 2014: POODLE (Padding Oracle On Downgraded Legacy Encryption)
 - 2015: FREAK (Factoring RSA Export Keys) (Downgrade RSA 512 bit)
 - 2015: Logjam-Angriff (Downgrade 512 Bit DH)
- ...



Some examples for attacks on cryptographic primitives

- 1998: DES (Data Encryption Standard) 56 bit key, EFF “Deep crack”
- ~2005: RC4, especially vulnerable when the beginning of the output keystream is not discarded, or when non-random or related keys are used (e.g. WEP)
- 2009: “MD5 considered harmful today: Creating a rogue CA certificate”
- 2013: Dual_EC_DRBG << backdoor
- 2018: IOTA Curl Hash function
- 2018: “Burning Zerocoins for Fun and Profit”
- 2019: SwissPost e-voting
- 2019: Zcash Counterfeiting Vulnerability
- ...



Observations

- *Established* cryptographic algorithms usually *not* fail abruptly but gradually (e.g. MD5, DES, SHA1)
 - Addendum: Unless they are not backdoored in the first place (e.g. Dual_EC_DRBG)
- *New/unstandardized* cryptographic algorithms *might* fail abruptly (e.g., Curl hash function in IOTA, Zcash, SwissPost e-voting, ...)
- But in the *majority of cases* the **implementation or usage is the problem!**
- => We need better **Cryptographic/Security Engineering**
Therefore we need to understand the problems first!



Fundamentals of Applied Cryptography



Cryptographic goals

- Some cryptographic goals:
 - confidentiality
 - integrity
 - authentication
 - non-repudiation
 - repudiation/deniability
 - Zero-knowledgeness
 - forward secrecy
 - privacy
 - unpredictability
 - ...
- Often more than one cryptographic goal is desired
 - e.g., it is not enough to have an encrypted communication (confidentiality) channel when the modification (integrity) of messages is a problem
- To design *secure systems* you might need to achieve even more goals that cannot solely be addressed by cryptography alone:
 - availability, liveness, agreement, termination, safety properties, ...



Important cryptographic goals

Confidentiality

- protect content from unauthorized readers
 - message can not be read
 - e.g. AES encryption of one block in ECB mode

Integrity

- protect content from unauthorized alteration/manipulation
 - message cannot be changed
 - e.g. MAC (Message authentication code) with shared group key

Authentication

- identification of data or communicating entities
 - message sender is identifiable
 - e.g., MAC (Message authentication code) with key per participant

[1] <http://cacr.uwaterloo.ca/hac/>

Important cryptographic goals

Non-repudiation

- prevent entity from denying previous commitments or actions
 - once a message has been received, it cannot be denied that this message has been sent
 - e.g., digital signature algorithm (ECDSA, EdDSA, RSA, ...)

Repudiation (deniability)

- enable entity to deny previous commitments or actions
 - it can be plausibly denied that this message has been sent by the sender
 - e.g., OTR (Off-the-Record messaging), MAC where key is released in follow-up message

[1] <http://cacr.uwaterloo.ca/hac/>

Important cryptographic goals

Authenticated Encryption (AE)

- sometimes also AEAD (authenticated encryption with additional data)
 - additional data is not confidential, but the integrity is protected
- since around 2000, the “go-to” way to do encryption
- confidentiality + integrity + authentication
 - e.g. AES encryption GCM mode, CCM mode, ...

(Perfect) Forward Secrecy (P)FS

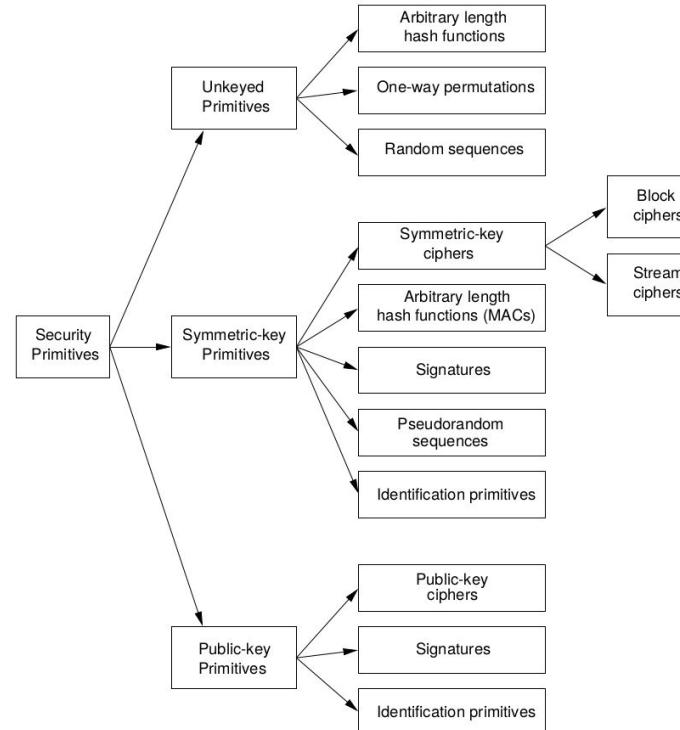
- past sessions are protected against future compromise / leakage of the long term asymmetric keys used by the communication parties does not enable decryption of previously transmitted data
 - recorded encrypted message should not be decryptable if long term asymmetric keys are compromised later
 - e.g., generate different (ephemeral) keys for each session (DH, ECDH)

[1] <http://cacr.uwaterloo.ca/hac/>



Important cryptographic primitives

- **Unkeyed primitives**
 - hash functions
 - (real) random sequences
- **Symmetric-key primitives**
 - symmetric key ciphers
 - block ciphers
 - stream ciphers
 - Message authentication codes
 - signatures
 - pseudo-random sequences
- **Public-key primitives**
 - public-key ciphers
 - signatures



Security of cryptographic primitives

- **Kerckhoffs' principle**
 - “A cryptosystem should be secure even if everything about the system, except the key, is public knowledge” [1]
 - Opposite of *security through obscurity*
- **“Schneier’s Law”**
 - “any person can invent a security system so clever that she or he can't think of how to break it.” [2]
 - That is why the algorithm/design should be public to enable evaluation and testing by others

[1] https://en.wikipedia.org/wiki/Kerckhoffs's_principle

[2] https://www.schneier.com/blog/archives/2011/04/schneiers_law.html



Security of cryptographic primitives

In cryptography, the following types high level types of security guarantees can be distinguished:

- **Information Theoretic Security**
 - sometimes also referred to as *unconditional security*, or in context of encryption *perfect secrecy*
 - secure against **computationally unbounded** adversary
 - Every possible solution is equally probable if the key is unknown
 - Examples:
 - *One time pad* (keys must be random and as long as the message)
 - *Shamir secret sharing* (as long as threshold t out of n not reached)
 - *Pedersen commitments*
 - Does not mean scheme is secure against any attack!
 - e.g., integrity is still an issue for the *one time pad* and Shamir secret sharing (SSS)

[1] <http://cacr.uwaterloo.ca/hac/>



Security of Cryptographic Primitives

- **Computational Security**
 - sometimes also referred to as *conditional security*
 - secure against a **computationally bounded** adversary
 - Algorithms e.g., Block- and stream ciphers (DES, AES, ...)
 - No feasible attack found, but there may be better attacks
 - **Provable Security**
 - Also requires computationally bounded adversary
 - Additionally, a mathematical proof (reduction) exists that breaking the scheme is as hard as solving some other problem that is assumed to be hard
 - Assume that we can break the security of A , then this implies that we can also break the security of B , but B is assumed to be hard so A must be hard as well
 - Algorithms e.g., RSA, DH, Elgamal, ... Problems e.g., factoring, discrete log, CDH, DDH, ...
 - proofs generally do not consider side-channel attacks or other implementation-specific attacks

Attack model

- Describe the capabilities of the attacker the respective system, or cryptographic primitive, is designed to protect against
- In context of cryptography, there are different attack models to consider depending on the concrete cryptographic primitive which should be designed

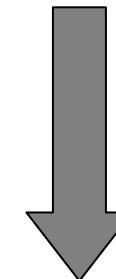


Attack models for encryption

Different attack models against encryption (key is assumed to be unknown):

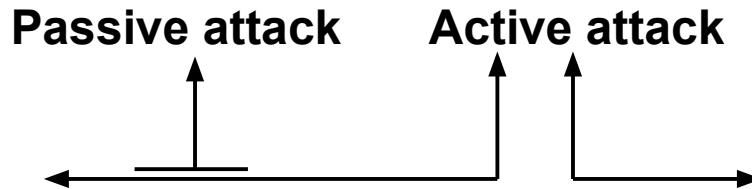
- Ciphertext only (COA)
 - attacker only knows c
 - e.g: *attacker intercepts encrypted message*
- Known plaintext (KPA)
 - attacker knows m and c
 - e.g: *attacker can obtain some message,cipher pairs*
- Chosen plaintext (CPA)
 - If achieved, also referred to as *semantically secure* under chosen plaintext attack
 - attacker can choose m and obtain c
 - e.g: *attacker has an encryption black box*
- Chosen ciphertext (CCA)
 - attacker can choose some c and obtain the corresponding m
 - e.g: *attacker has a decryption black box for some messages but not for the message he is up to break (otherwise it would be trivial of course)*

More powerful attack



Attack models for communication

- Important to differentiate between **active** and **passive** attacks. Passive attackers just observe - active attackers can interfere at will with messages (manipulate, delay, resend, ...).



- If we want to answer the question “*is this secure?*” then the answer always depends on the used attack/threat model and required cryptographic/security goals!
 - Can integrity be ensured against a *passive* attacker that only has access to the ciphertexts (COA)? Answer might be **yes**
 - Can integrity be ensured against an *active* attacker that only has access to the ciphertexts (COA)? Answer might be **no**

Symmetric-Key Cryptography



universität
wien

SBA
Research

Encryption



universität
wien

SBA
Research

Encryption

\mathcal{A} **Alphabet** of definition (not further discussed here, we assume binary)

- finite set of symbols, e.g., binary alphabet $\mathcal{A} = \{0, 1\}$

\mathcal{M} **Message space**

- set that contains strings from symbols of an alphabet
- elements of \mathcal{M} are called **plaintext** messages or just **messages** $m \in \mathcal{M}$

\mathcal{C} **Ciphertext space**

- set that contains strings from symbols of an alphabet
- elements of \mathcal{C} are called **ciphertext** messages $c \in \mathcal{C}$

\mathcal{K} **Key space**

- Elements of \mathcal{K} are called **keys** $k \in \mathcal{K}$
- For all practical encryption functions the key length can be smaller than the message length.



Symmetric-key encryption

A computationally secure symmetric cipher $\mathcal{E} = (E, D)$, where E and D are efficient algorithms, is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Each element $k \in K$:

- uniquely determines a bijective mapping $E_k : M \rightarrow C$
- uniquely determines a bijective mapping $D_k : C \rightarrow M$

$$c = E(k, m)$$

$$m = D(k, c)$$

$$\forall m \in M, k \in K : m = D(k, E(k, m))$$

Symmetric-key encryption

- **Block ciphers**
 - break up plaintext into strings (blocks) of fixed length
 - Encrypt/decrypt one block at a time
 - uses *substitution* and *transposition (permutation)* techniques
 - e.g.: AES, DES, ...
 - Modes **AES-GCM, AES-CCM, (AES-OCB)**
- **Stream ciphers**
 - Special case of block cipher with length **1**
 - however, substitution technique can change for every block
 - Key stream $\{k[0], k[1], k[2], \dots\}$
 - e.g.: RC4, **ChaCha20-Poly1305**, ...



Block Ciphers



universität
wien

SBA
Research

Modes of operation for Block Ciphers

- Block cipher encrypts blocks of fixed size
 - DES: 56 bit key, 64 bit block
 - AES: 128, 192, or 256 key, 128 bit block
 - ...
- Modes of operation for Block Ciphers
 - ~~– Electronic Code Book (ECB)~~
 - ~~– Cipher Block Chaining (CBC)~~
 - ~~– Counter Mode (CTR)~~
 - Galois Counter Mode (GCM)**
 - Counter with CBC-MAC (CCM)**
 - Offset Codebook Mode (OCB)**

Authenticated
Encryption
(AE)



This is what you
want to use!



universität
wien

SBA
Research

Our attack model

We pick between the following capabilities of an attacker depending on our example:

- If not stated differently, for all our examples it holds true that, the victim/target uses the **same** (unknown) **secret** key k in all steps. The secret key is long enough and chosen uniformly at random (i.e., sampled from a set where drawing each element is equally probable e.g., $\{0,1\}$ 50% each)
- **Passive attacker:** Can observe ciphertexts (e.g. listen to network traffic)
- **Active attacker:** Can observe and manipulate ciphertext (e.g., Man-in-the-middle attacks)
- **COA** (Ciphertext Only Attack): Attacker only knows ciphertext
- **KPA** (Known Plaintext Attack): Attacker knows some plaintexts, ciphertext pairs. We also assume that the attacker knows the structure of the messages but not the concrete values in all cases.
 - For example he knows that messages have the form of
 - | command | value | username |



ECB



universität
wien

SBA
Research

ECB

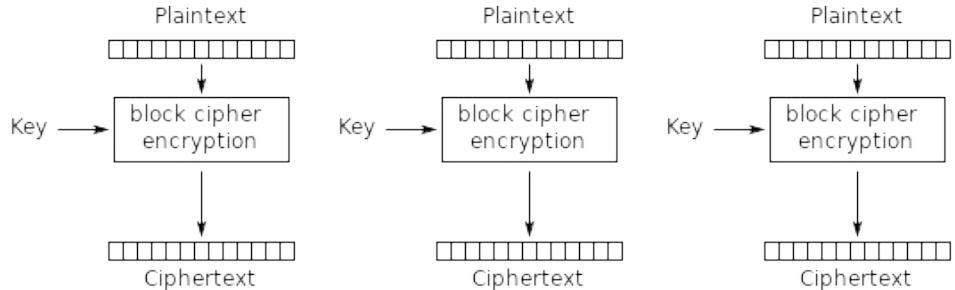
- Electronic Code Book (ECB)
- Split message into blocks
- Pad message with random data so its length is a multiple of the block size
- Feed every block separately into the encryption function $E(k, m[i])$ (e.g. AES) to encrypt the message



ECB

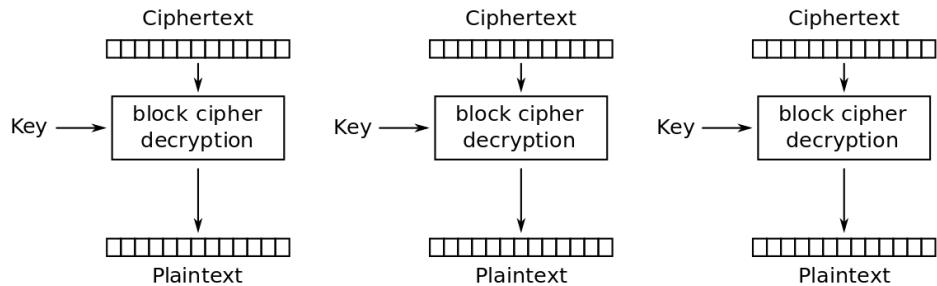
Encryption and decryption

Enc. :



Electronic Codebook (ECB) mode encryption

Dec. :



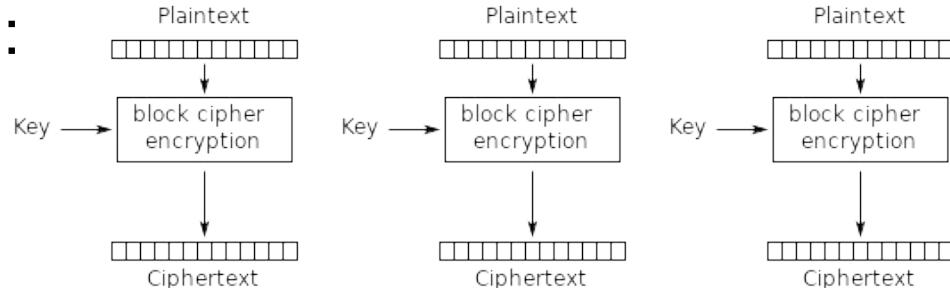
- [*] wikipedia.org

Electronic Codebook (ECB) mode decryption

ECB

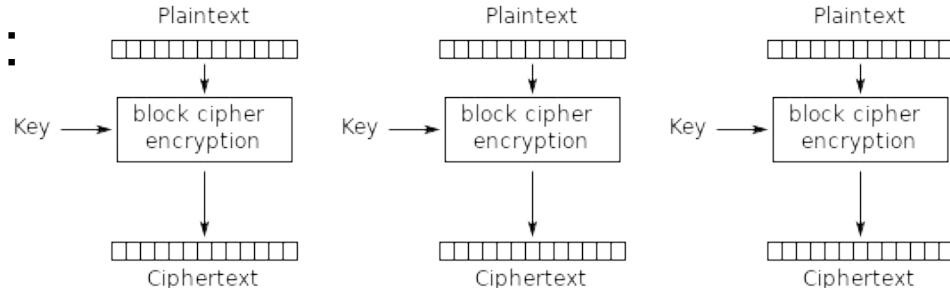
What potential information might leak? (passive attack; COA)

Run 1:



Electronic Codebook (ECB) mode encryption

Run 2:



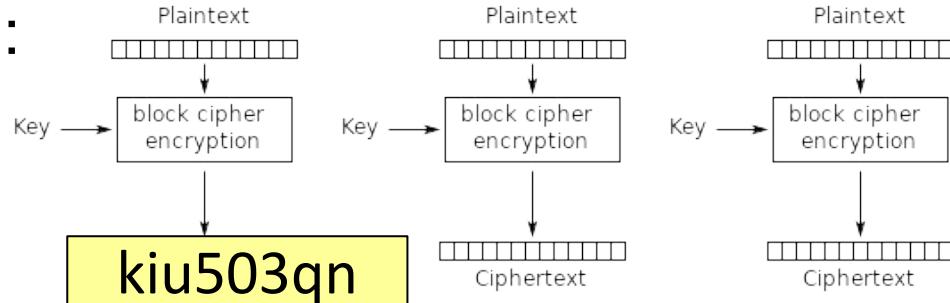
- [*] wikipedia.org

Electronic Codebook (ECB) mode encryption

ECB

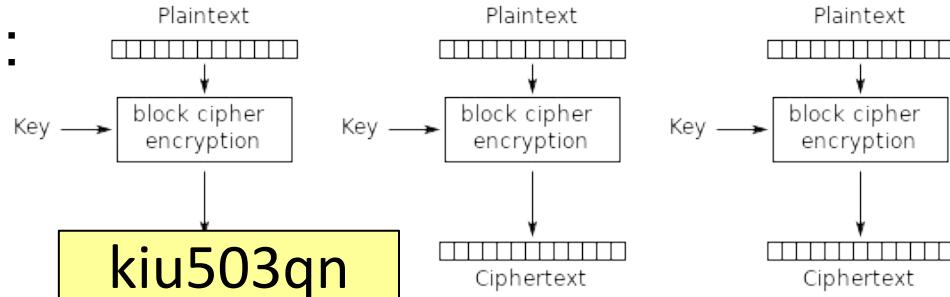
What potential information might leak? (passive attack; COA)

Run 1:



Electronic Codebook (ECB) mode encryption

Run 2:



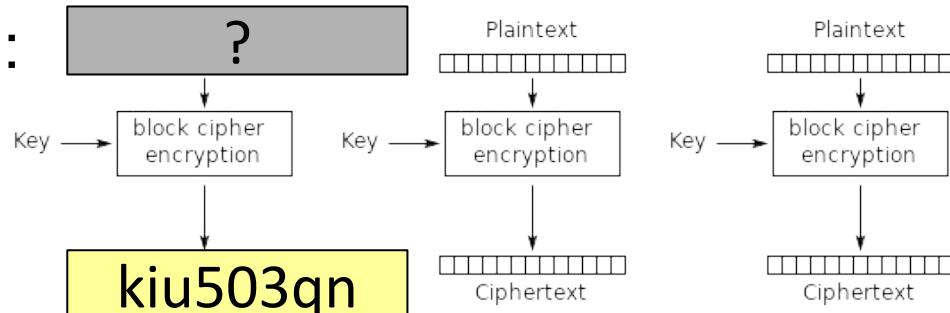
- [*] wikipedia.org

Electronic Codebook (ECB) mode encryption

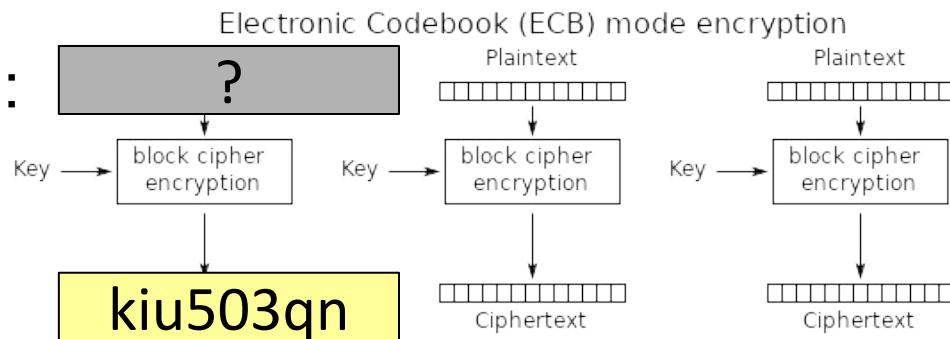
ECB

Same message was sent again + metadata! (passive; COA)

Run 1:



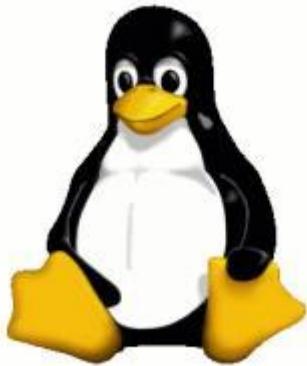
Run 2:



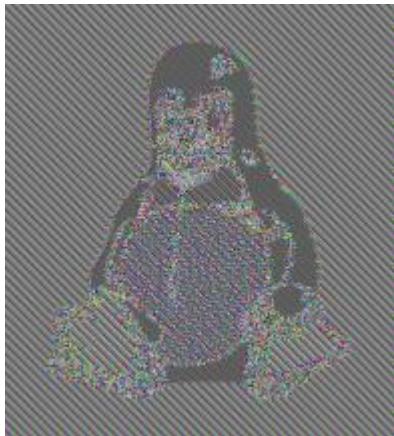
- [*] wikipedia.org

Electronic Codebook (ECB) mode encryption

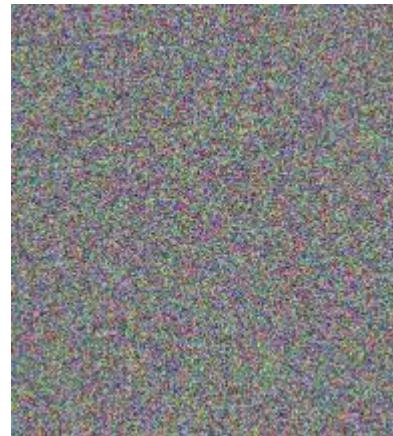
ECB



Plain-text



ECB



CBC

- [*] wikipedia.org



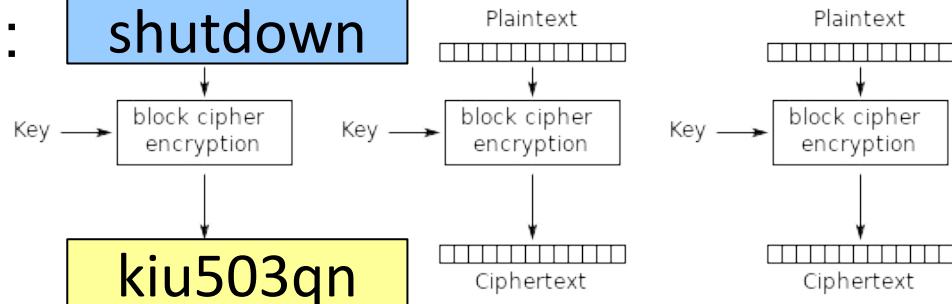
universität
wien

SBA
Research

ECB

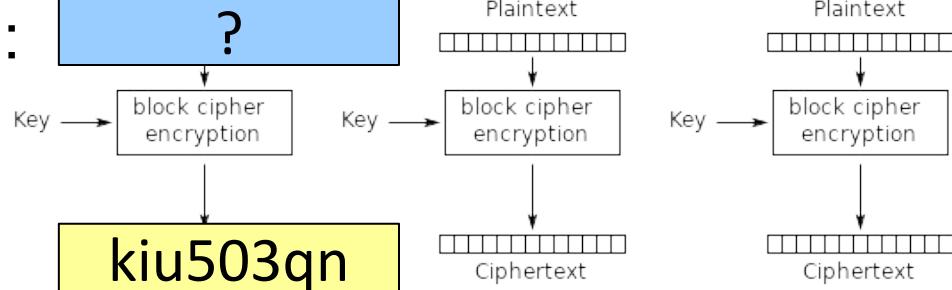
What is the message ‘?’ (passive; KPA)

Run 1: **shutdown**



Electronic Codebook (ECB) mode encryption

Run 2:



Electronic Codebook (ECB) mode encryption

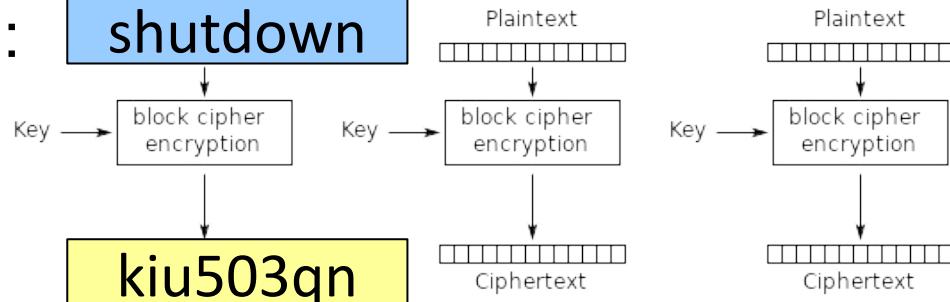
- [*] wikipedia.org

ECB

Under same key!

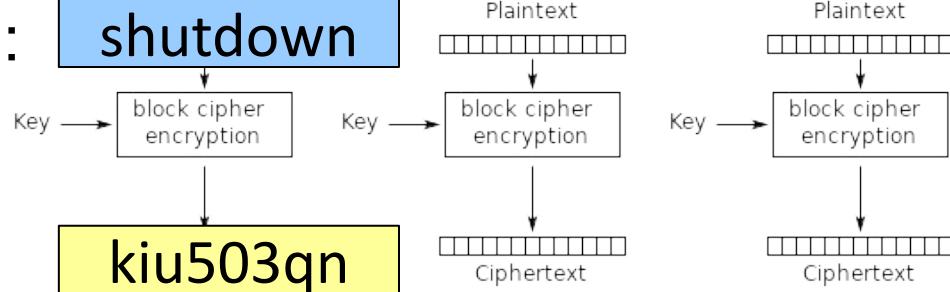
If plaintext known once => always detectable! (passive; KPA)

Run 1: **shutdown**



Electronic Codebook (ECB) mode encryption

Run 2: **shutdown**

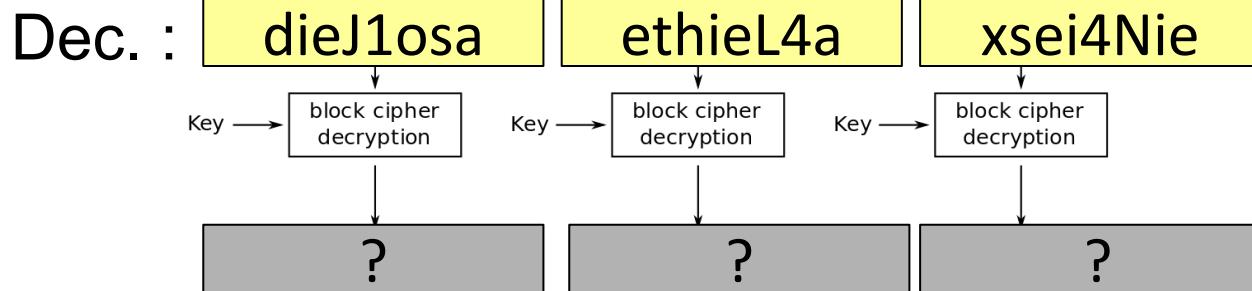
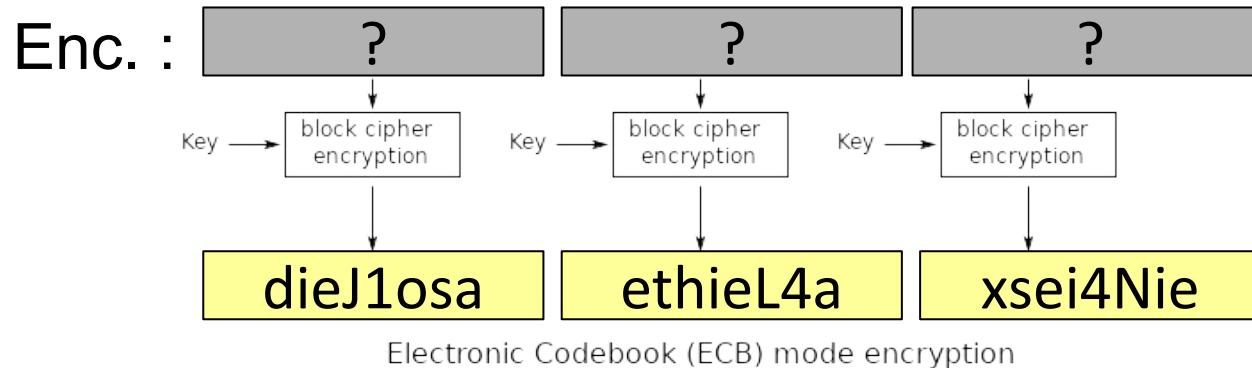


- [*] wikipedia.org

Electronic Codebook (ECB) mode encryption

ECB

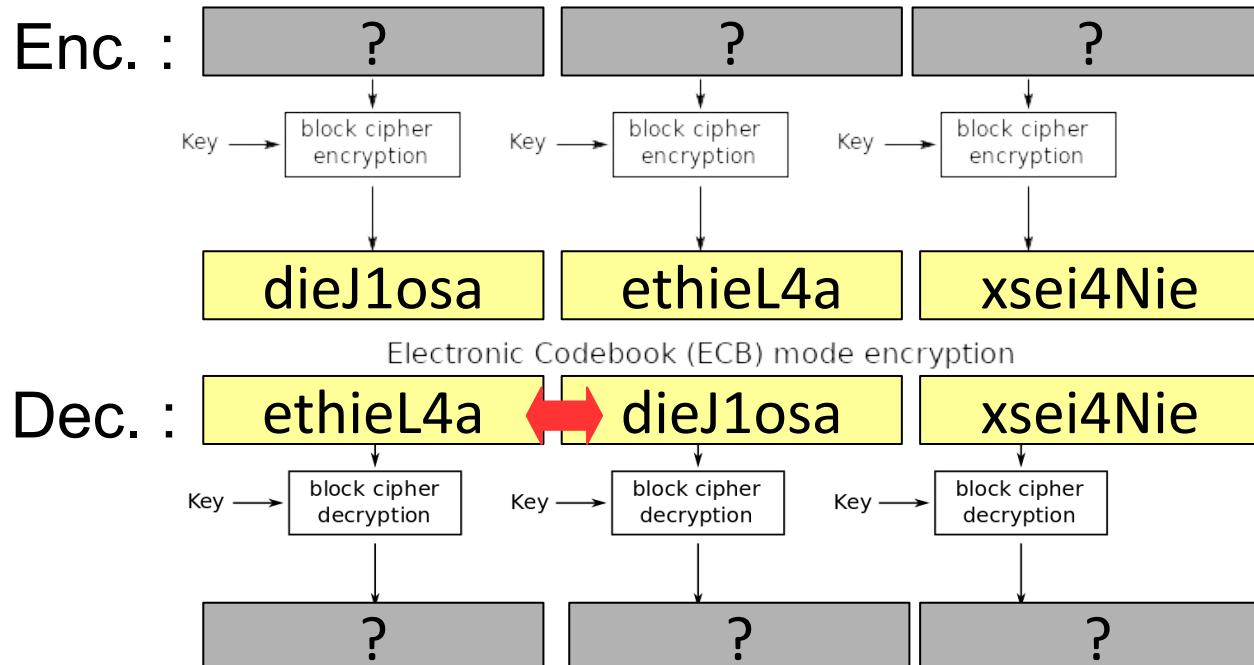
What active attack is possible? (active; COA)



- [*] wikipedia.org

ECB

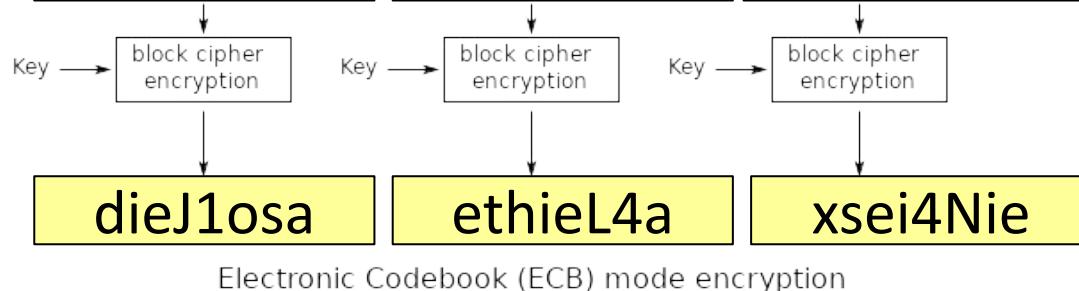
Attacker can change ordering, drop & replay! (active; COA)



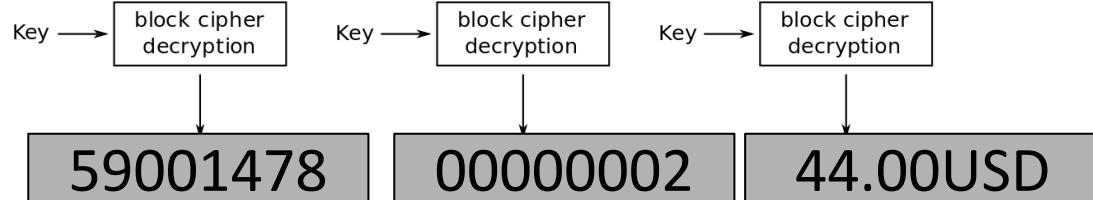
ECB

Attacker can change ordering, drop & replay! (active; COA)

Enc. : **00000002** **59001478** **44.00USD**



Dec. : **ethieL4a** **dieJ1osa** **xsei4Nie**



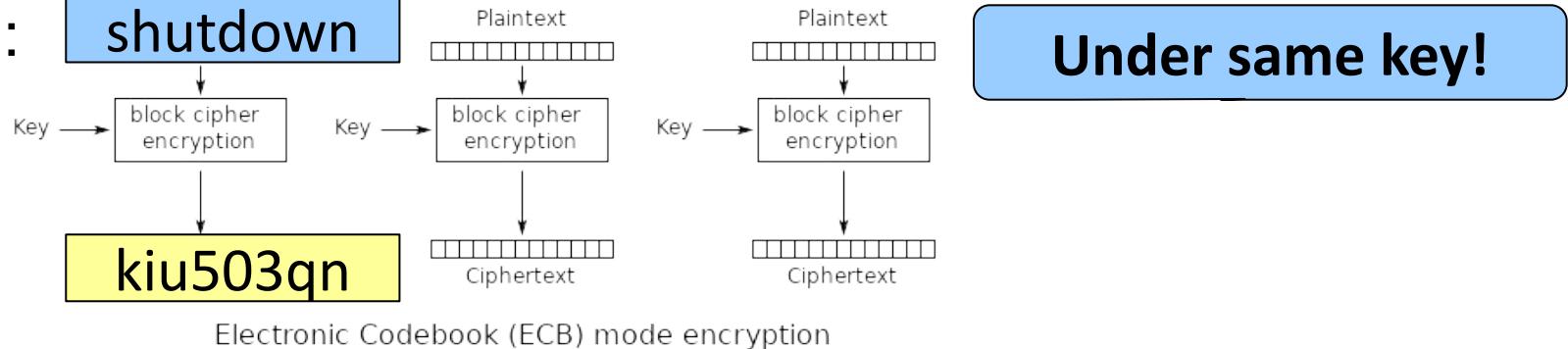
- [*] wikipedia.org

Electronic Codebook (ECB) mode decryption

ECB

What other active attack is possible (active; KPA)

Run 1: **shutdown**



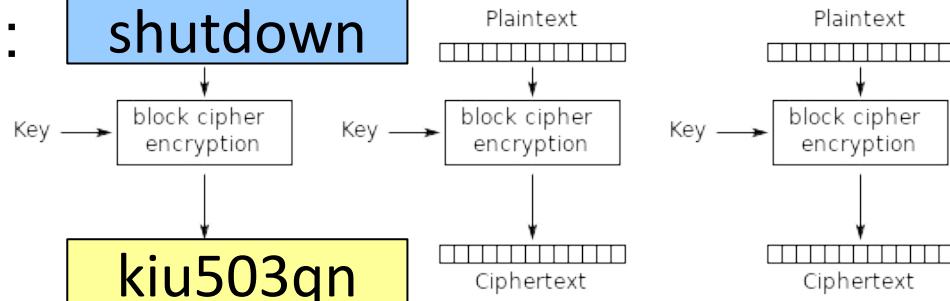
Run 2:

- [*] wikipedia.org

ECB

What other active attack is possible (active; KPA)

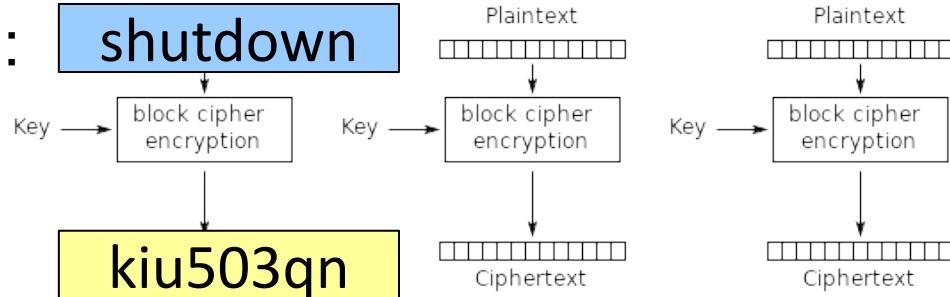
Run 1: **shutdown**



Under same key!

Electronic Codebook (ECB) mode encryption

Run 2: **shutdown**



Replay attack

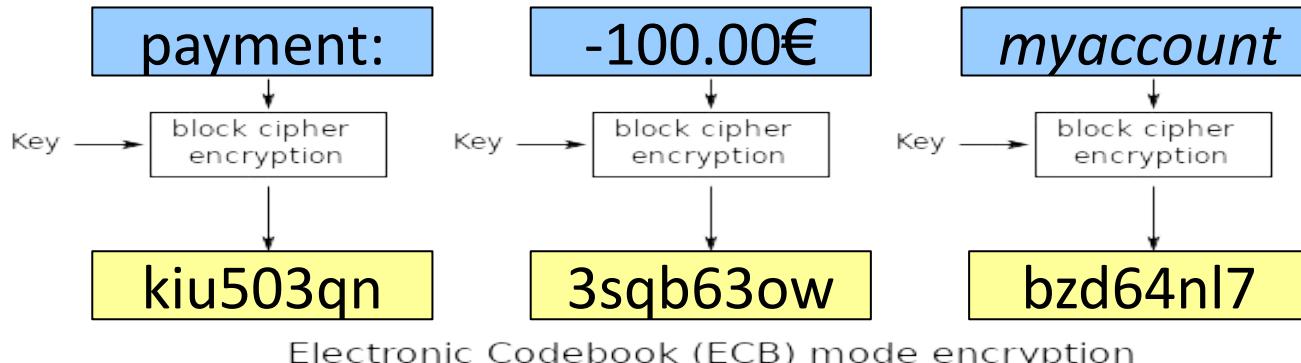
Electronic Codebook (ECB) mode encryption

- [*] wikipedia.org

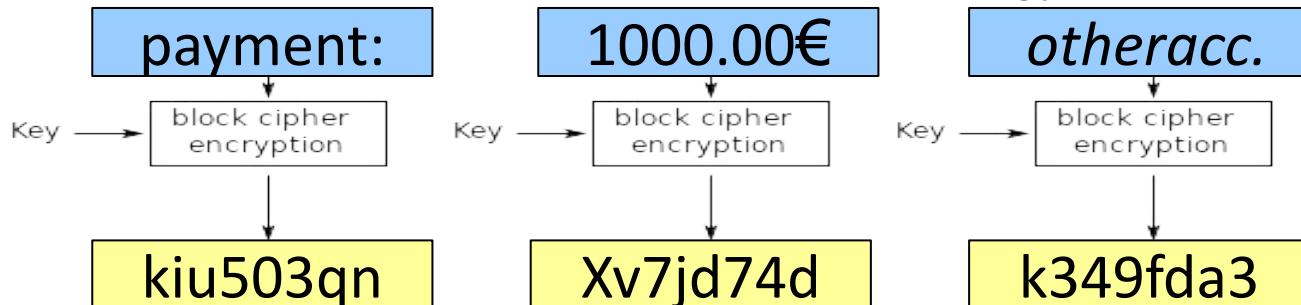
ECB

Active attack easier when KPA possible! (active; KPA)

Run 1:



Run 2:



- [*] wikipedia.org

ECB

Change known blocks to reach goal (active; KPA)

Generate **payment:**

Run 3:

↓
1000.00€

↓
myaccount

↓
kiu503qn

↓
Xv7jd74d

↓
bzd64nl7

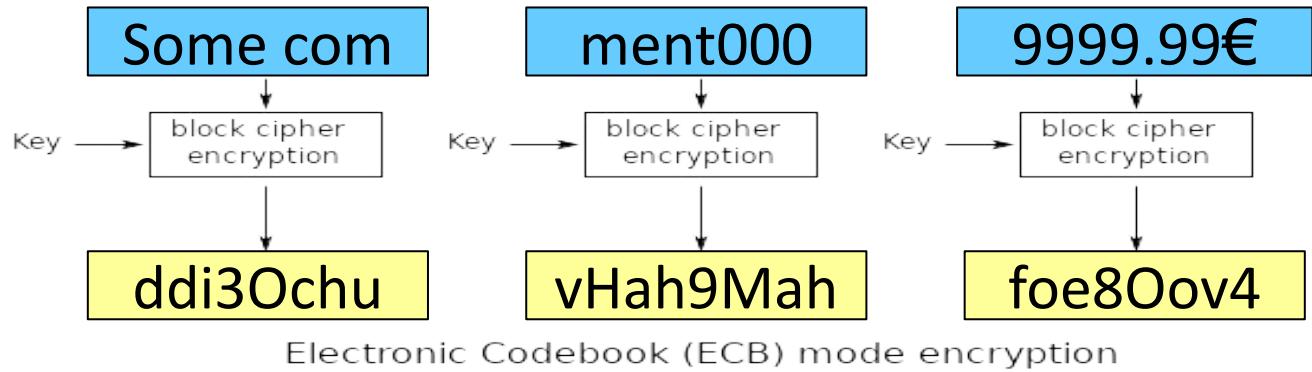
Electronic Codebook (ECB) mode encryption

- [*] wikipedia.org

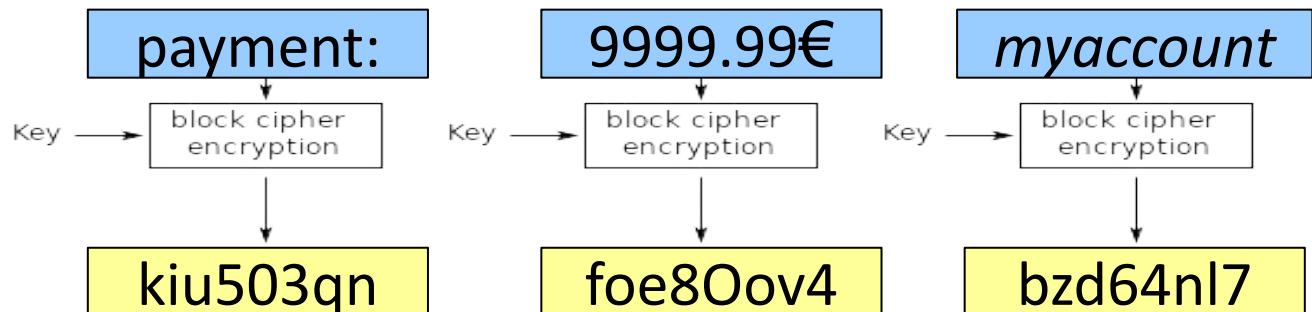
ECB

Active attack easier when CPA possible (active; CPA)

Construct message block with desired value e.g., by adding comment s.t.



Construct message s.t.



ECB

Potential Problems:

- Each block encrypted independently of other blocks
 - ECB does not hide data patterns (repetitions)
 - Under the same key **same messages/plaintexts** result in the **same ciphertexts**
- Vulnerable to block insertion and deletion
 - Attacker can combine and **reorder** individual blocks from different messages into a new message (under the same key)
 - **Replay** attacks



Defenses

- Don't use ECB!
- Use AE (Authenticated Encryption) modes
 - e.g., GCM, CCM, OCB
- See part on message authentication codes (MAC) for more information.



ECB

- Who uses ECB anyway?
 - German “Staatstrojaner” in its 2011 version [1,2]
 - “*... ein symmetrisches Verfahren (AES) im nicht-verketteten Placebo-Modus ECB verwendet ...*
 - *Der Schlüssel ist zudem fest einprogrammiert.*” [1]
 - So they used ECB and a shared secret key among all “installations”



[1] <https://www.ccc.de/system/uploads/76/original/staatstrojaner-report23.pdf>

[2] <http://www.heise.de/security/meldung/CCC-kritisiert-neue-Staatstrojaner-Version-1366993.html>

CBC



universität
wien

SBA
Research

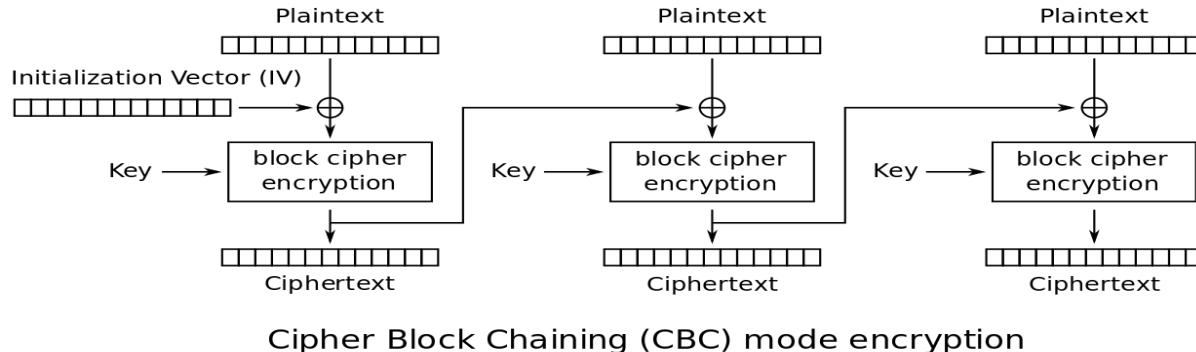
CBC

- Cipher block chaining mode (CBC)
- Split message into blocks
- Pad message with random data so its length is a multiple of the block size
- **Encryption of one block depends (also) on previous block**
- Encryption of first block depends (also) on a random Initialization Vector (IV) per message
- XOR every block with the cipher text of the last block before feeding it into the encryption function (e.g. AES)

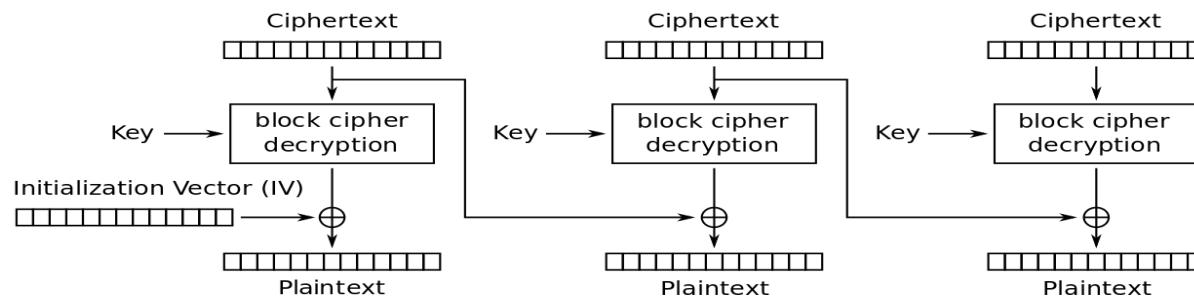


CBC

Enc. :



Dec. :



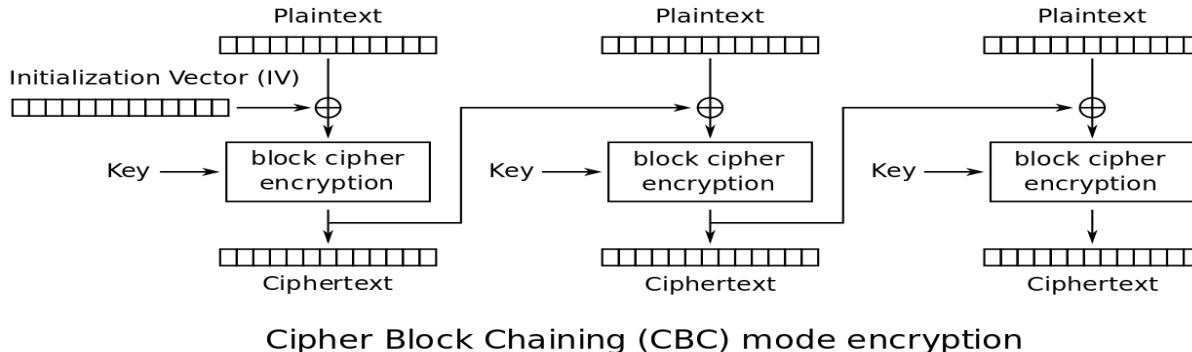
- [*] wikipedia.org

Cipher Block Chaining (CBC) mode decryption

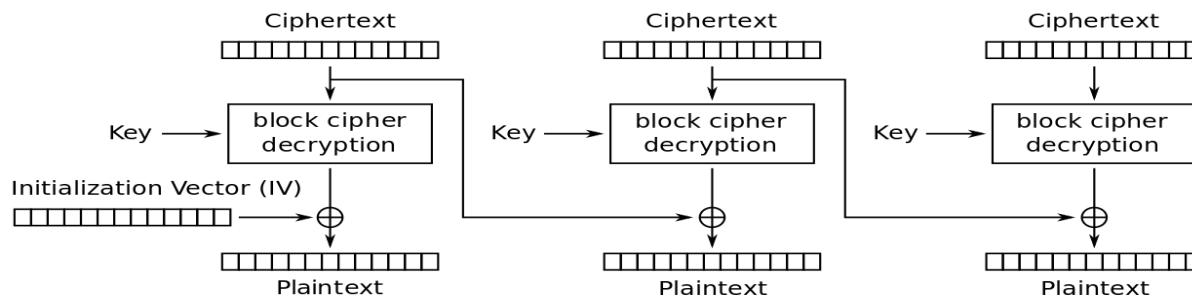
CBC

What happens if IV does not change? (passive attack; COA)

Enc. :



Dec. :

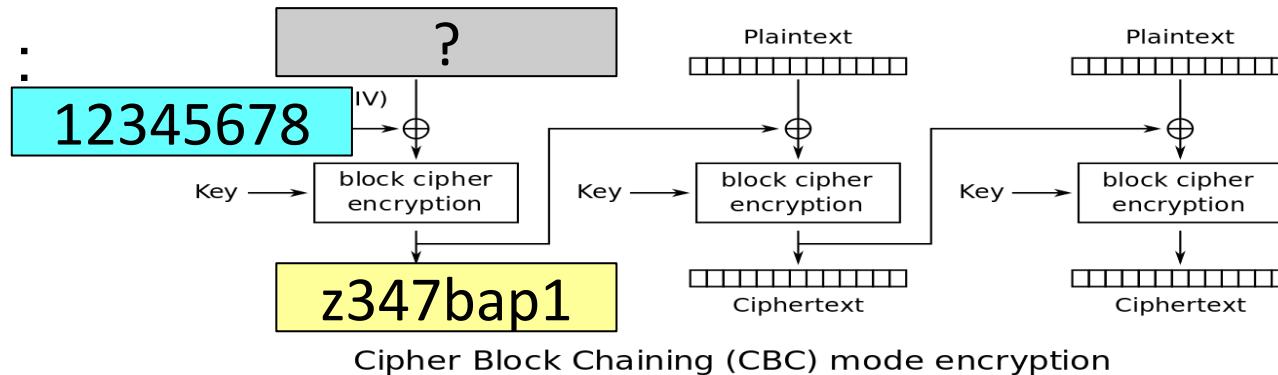


- [*] wikipedia.org

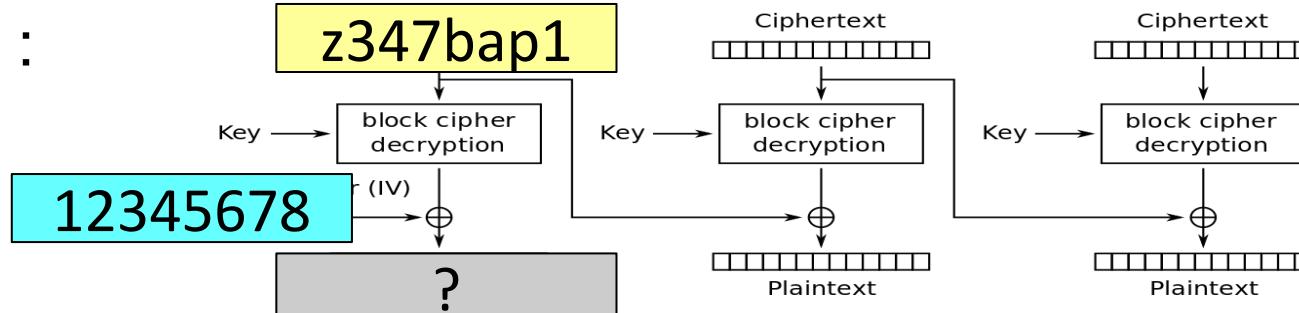
CBC

Same plaintext => same ciphertext! (passive; COA)

Enc. :



Dec. :

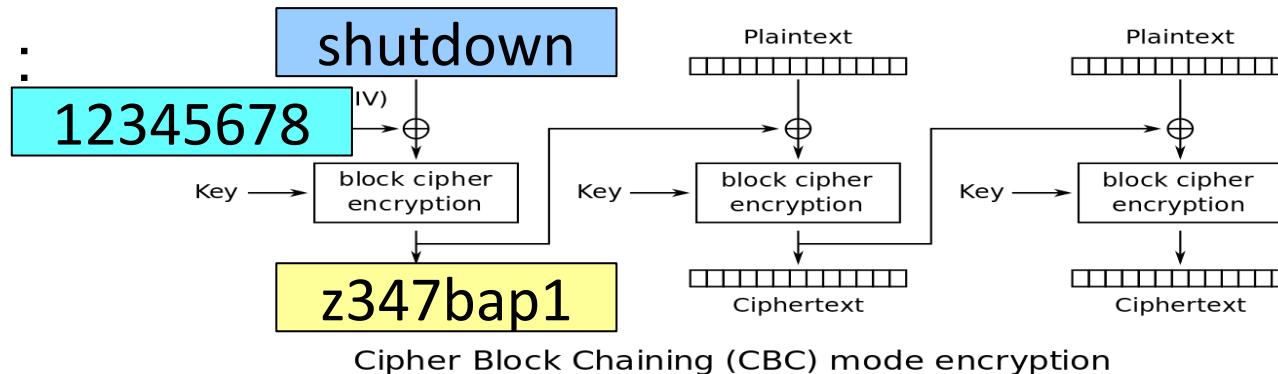


- [*] wikipedia.org

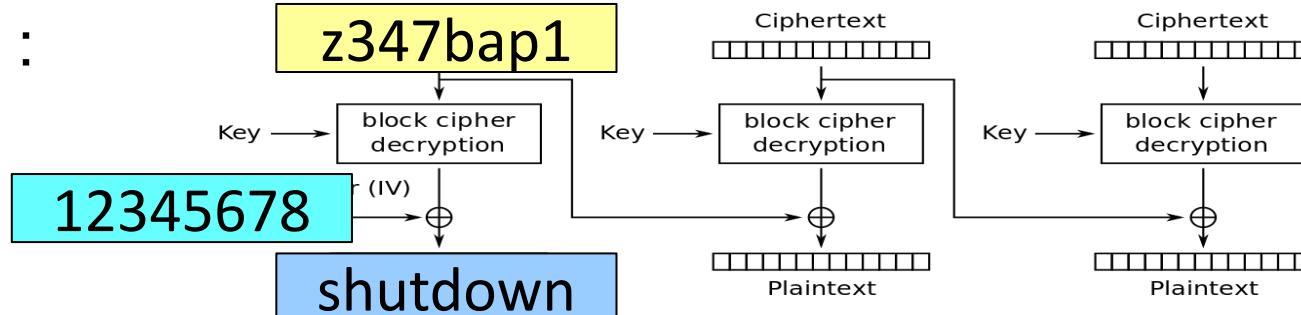
CBC

The same for KPA (passive attack; KPA)

Enc. :



Dec. :

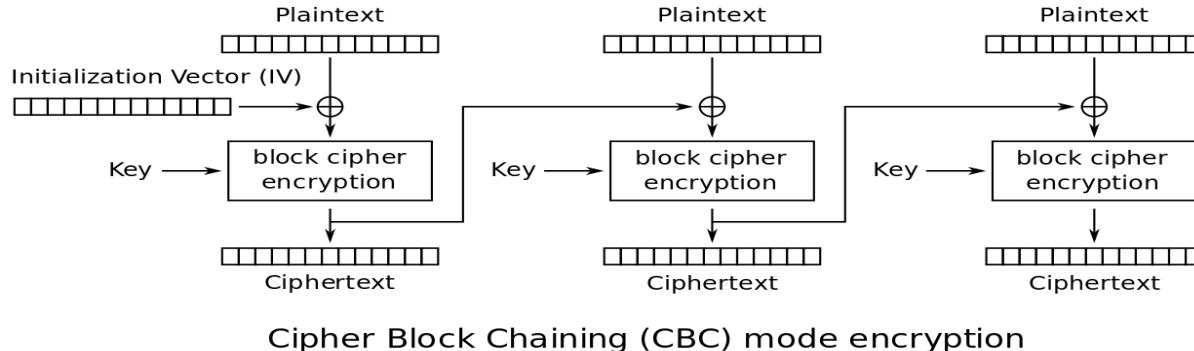


- [*] wikipedia.org

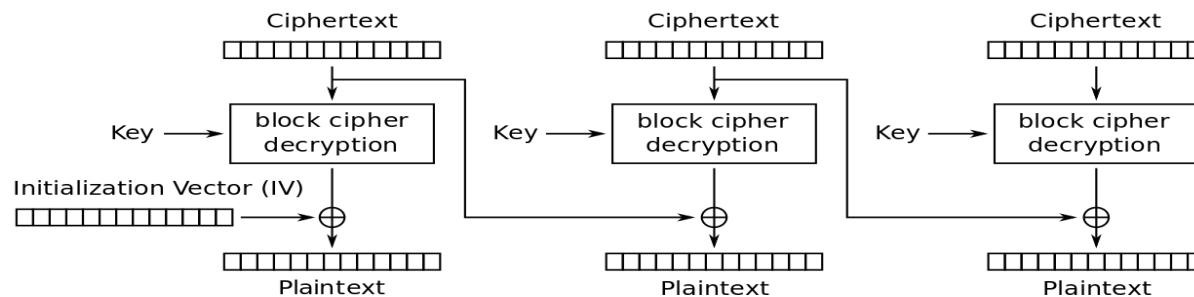
CBC

What other attacks are possible? (active;KPA)

Enc. :



Dec. :



- [*] wikipedia.org

Cipher Block Chaining (CBC) mode decryption

CBC

- CBC Bit Flipping Attack
 - The block containing the flipped byte will be mangled when decrypted, however the corresponding byte in the next decrypted block will be altered

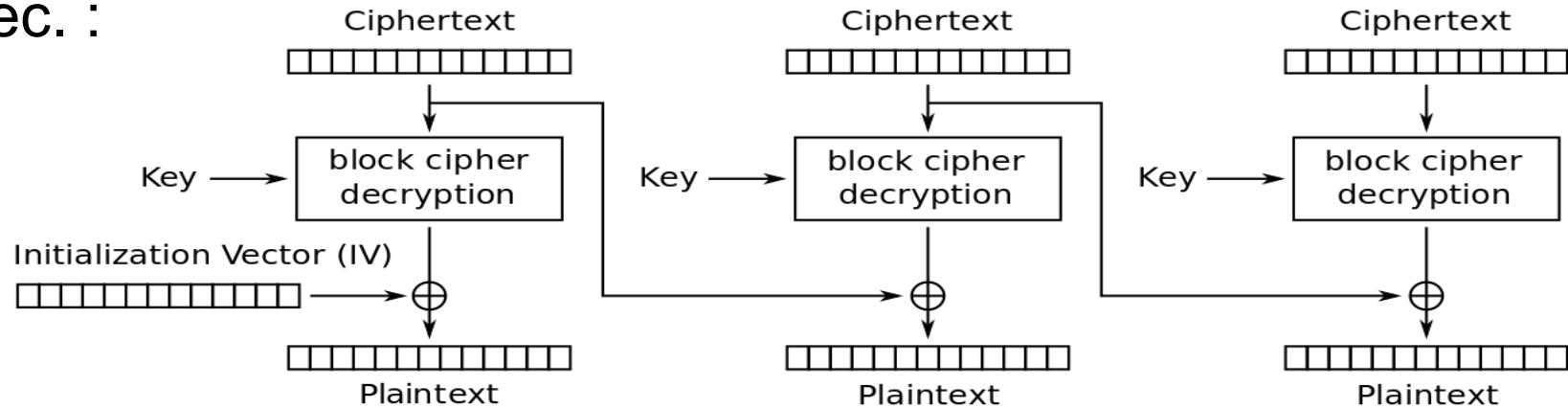
[1] <http://blog.gdssecurity.com/labs/2010/10/6/crypto-challenges-at-the-csaw-2010-application-ctf-qualifyin.html>



CBC

- CBC Bit Flipping Attack (active; KPA)

Dec. :



Cipher Block Chaining (CBC) mode decryption

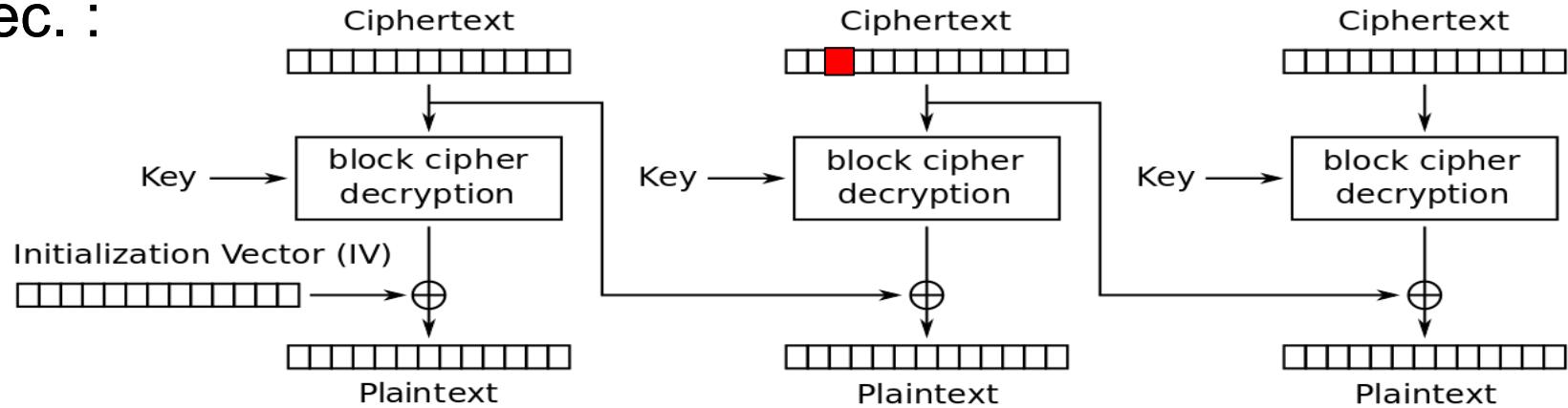
Plaintext :



CBC

- CBC Bit Flipping Attack (active; KPA)

Dec. :

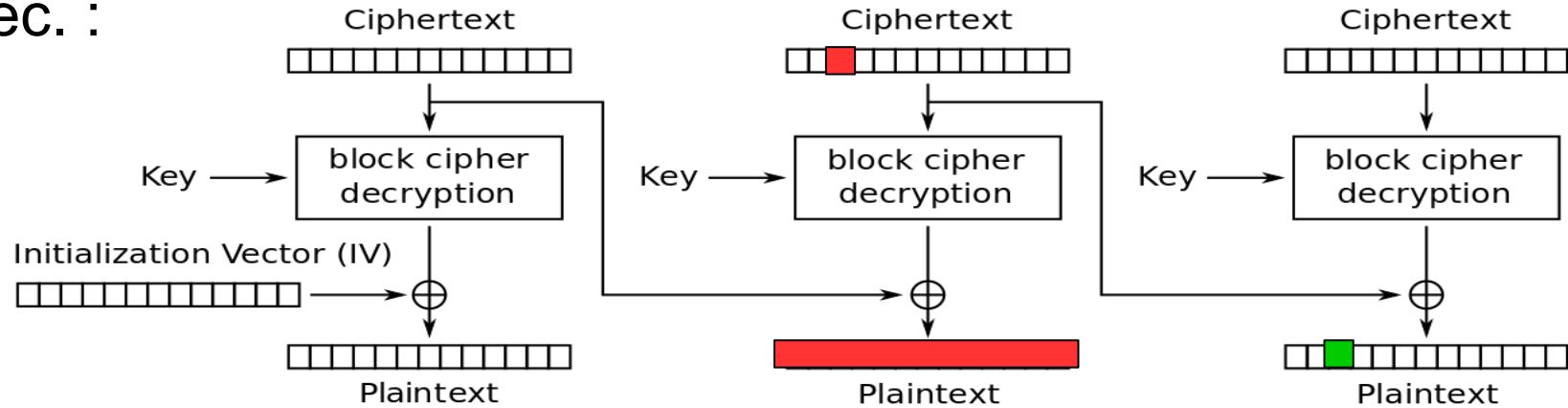


Ciphertext :

CBC

- CBC Bit Flipping Attack (active; KPA)

Dec. :

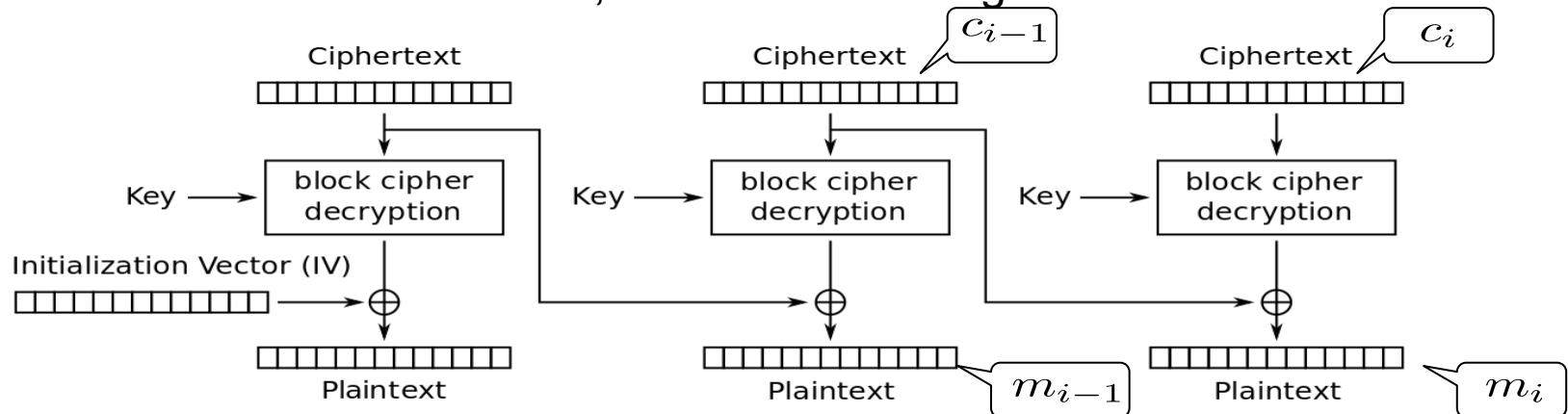


Cipher Block Chaining (CBC) mode decryption

Plaintext :

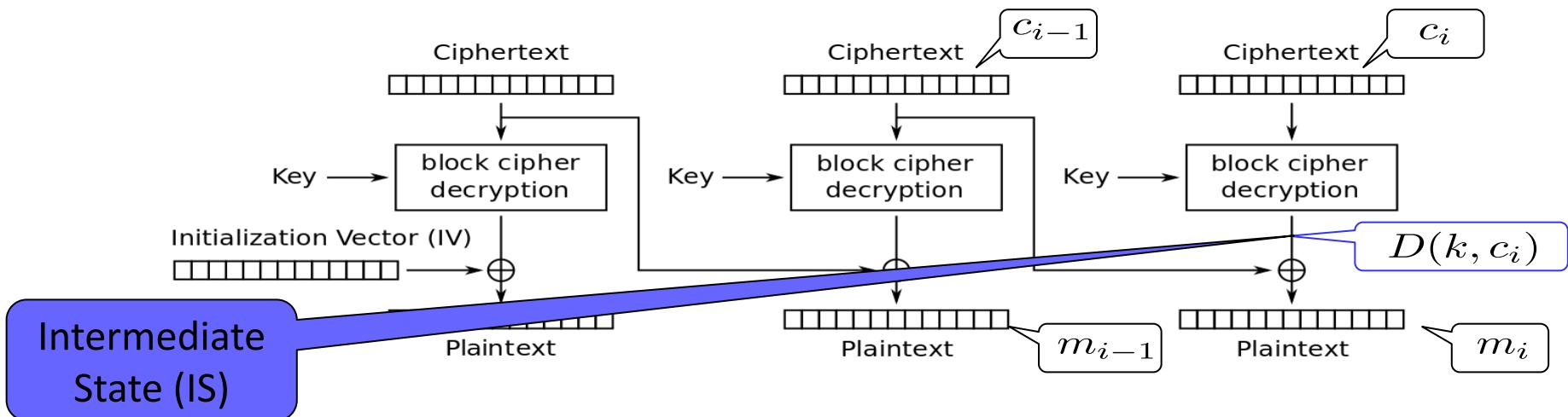
CBC

- CBC Bit Flipping Attack (active; KPA)
 - Ubuntu 12.04 full disk encryption CBC Bit Flipping Attack [1]
 - attacker knows m and c , but wants to change m



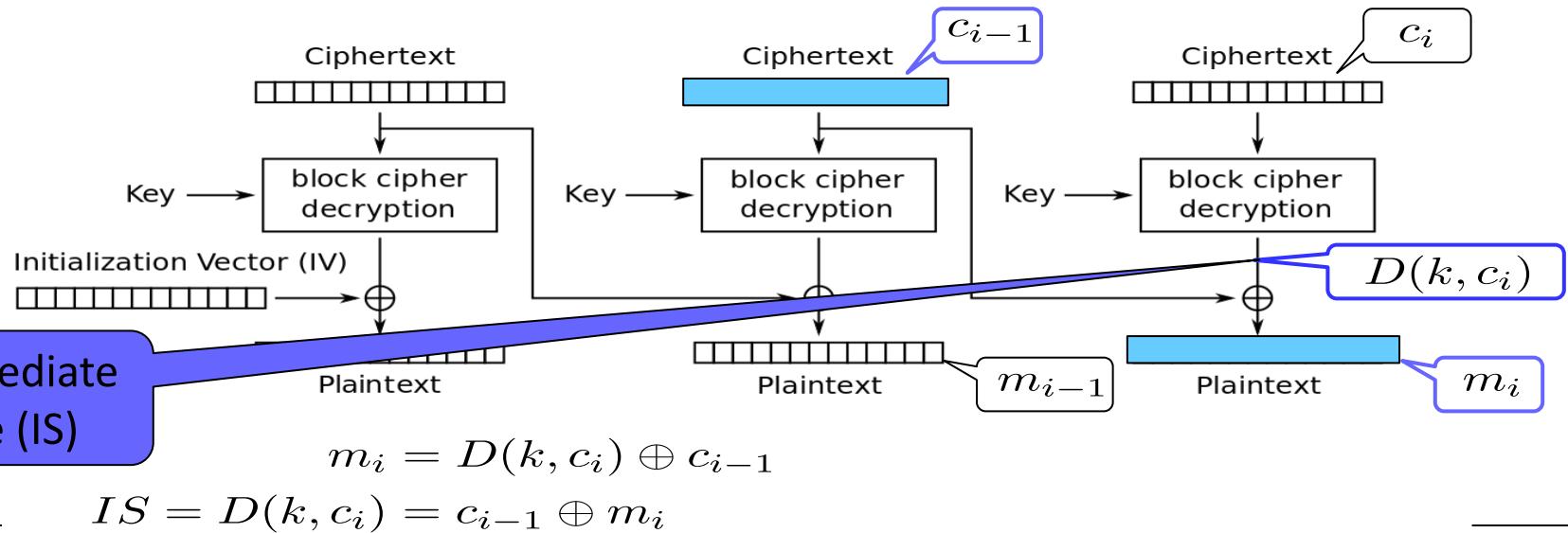
CBC

- CBC Bit Flipping Attack (active; KPA)
 - Ubuntu 12.04 full disk encryption CBC Bit Flipping Attack [1]



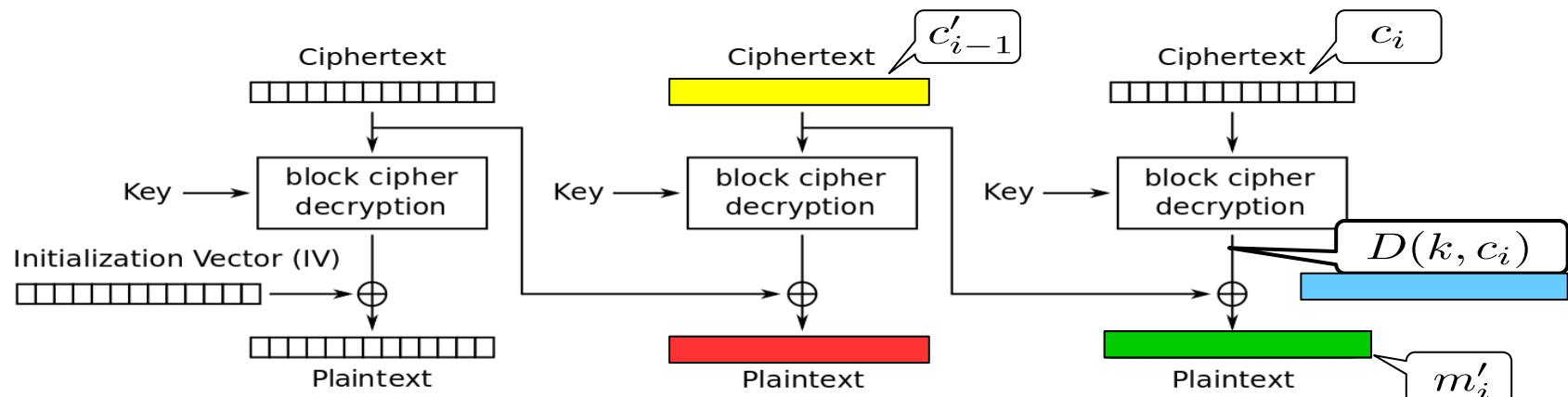
CBC

- CBC Bit Flipping Attack (active; KPA)
 - Ubuntu 12.04 full disk encryption CBC Bit Flipping Attack [1]



CBC

- CBC Bit Flipping Attack (active; KPA)
 - Ubuntu 12.04 full disk encryption CBC Bit Flipping Attack [1]



$$m_i = D(k, c_i) \oplus c_{i-1}$$

$$D(k, c_i) = c_{i-1} \oplus m_i$$

$$c'_{i-1} = D(k, c_i) \oplus m'_i$$

CBC

- Ubuntu 12.04 full disk encryption CBC Bit Flipping Attack [1]
 - Inject shellcode/commands in full disk encrypted default installation image without knowing decryption key
 - This can be done since *plaintext* and *ciphertext* is known which allows to reconstruct the intermediate state which then can be used in combination with the ciphertext to change the **plaintext of a follow up block**

The intermediate state (*IS*), is the direct output of the encryption function $D(k, c_i)$:

$$m_i = D(k, c_i) \oplus c_{i-1}$$

$$D(k, c_i) = c_{i-1} \oplus m_i$$

To change the plaintext to m'_i , the following equation can be used to calculate c'_{i-1} :

$$c'_{i-1} = D(k, c_i) \oplus m'_i$$

$$m'_i = c'_{i-1} \oplus D(k, c_i)$$

[1] <http://www.jakoblell.com/blog/2013/12/22/practical-malleability-attack-against-cbc-encrypted-luks-partitions/>

CBC

CBC Padding Oracle Attack (active; COA; side channel)

Padding:

- Block ciphers require that all messages are of the same well defined block length
- For the last block **padding** might be required to “fill” the missing input bytes
- One of the most common padding schemes is PKCS#5 padding.
- PKCS#5 padding is identical to PKCS#7 padding, except that it has only been defined for block ciphers that use a 64-bit (8-byte) block size.
- For our example we assume 8-byte block size since it is less to write ;)

[1] <http://blog.gdssecurity.com/labs/2010/10/6/crypto-challenges-at-the-csaw-2010-application-ctf-qualifyin.html>

CBC

CBC Padding Oracle Attack (active; COA; side channel)

- PKCS#5
 - In this scheme the **final block of plain text is always padded** with N bytes. Thereby, N depends on the length of the plain text in the last block:
 - A single byte padding (0x01)
 - Two bytes padding (0x02, 0x02)
 - Three bytes padding (0x03, 0x03, 0x03)
 - Four bytes padding (0x04, 0x04, 0x04, 0x04)

[1] <http://blog.gdssecurity.com/labs/2010/10/6/crypto-challenges-at-the-csaw-2010-application-ctf-qualifyin.html>

CBC

CBC Padding Oracle Attack (active; COA; side channel)

- CBC Padding example PKCS#5. there is always some padding

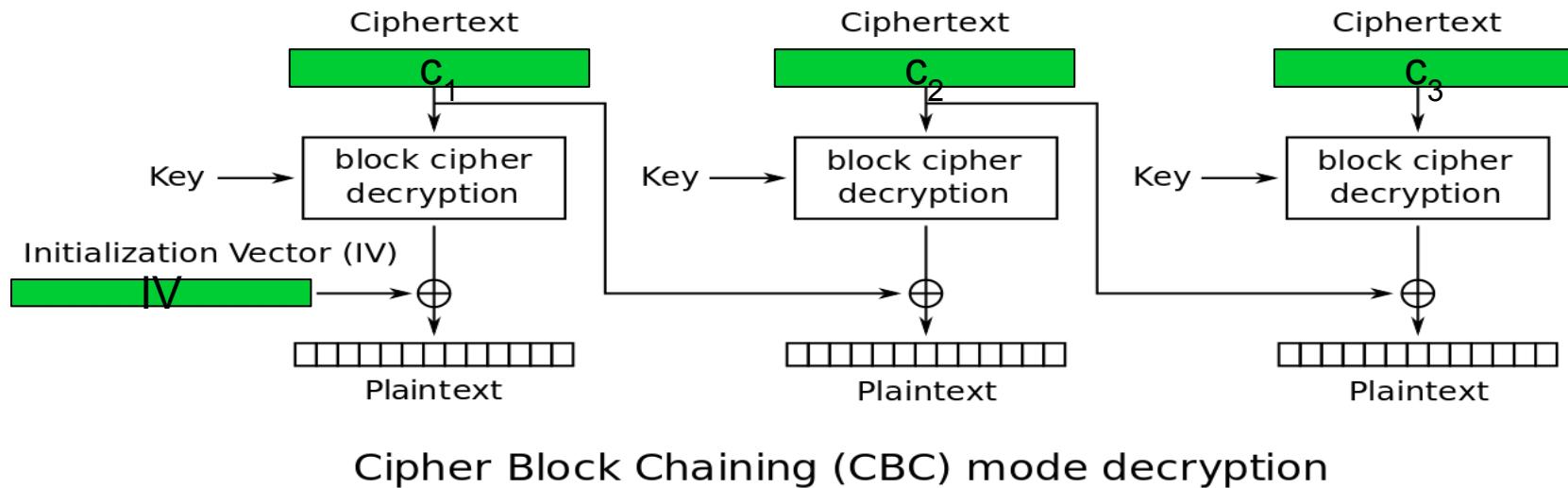
	BLOCK #1								BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04



CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Attacker knows cipher text and initialization vector and can feed both to the server to get a response $\{IV, c_1, c_2, \dots, c_n\}$



CBC

CBC Padding Oracle Attack (active; COA; side channel)

- CBC Padding oracle attack requires that the following cases can be distinguished:
 - When a **valid ciphertext** is received (one that is *properly padded* and contains valid data) the application responds normally
 - (e.g. 200 OK)
 - When an **invalid ciphertext** is received (one that, when decrypted, does *not end with valid padding*) the application throws a cryptographic exception
 - (e.g. 500 Internal Server Error)
 - When a **valid ciphertext** is received (one that is *properly padded*) but decrypts to an invalid value, the application displays a custom error message
 - (e.g. 200 OK, & Invalid Data)

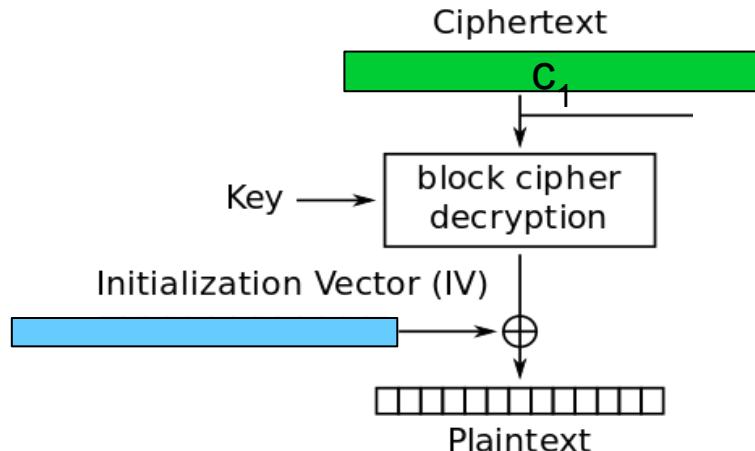
CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- Attacker sends initial cipher text block to server together with changed IV

	BLOCK 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D

INVALID PADDING X



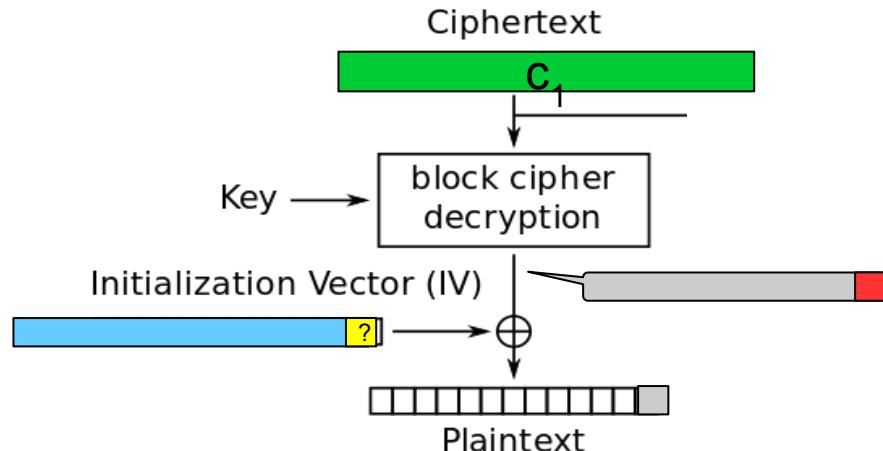
CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- Attacker brute forces IV until he does not receive a padding failure message

	BLOCK 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3C

INVALID PADDING X

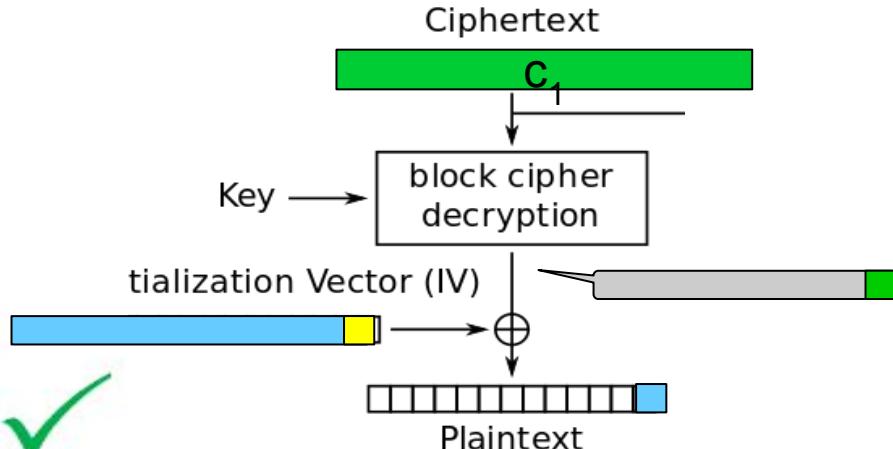


CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- If the IV produces the right padding byte at the end the padding is correct

	Block 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01
	VALID PADDING							



CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- Calculate the IS given the IV that produced the right padding as well as the padding

	Block 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xEC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01

VALID PADDING



The intermediary state byte 8, is calculated as follows:

$$IS[8] \oplus IV[8] = 0x01$$

$$IS[8] \oplus 0x3C = 0x01$$

$$IS[8] = 0x3C \oplus 0x01$$

$$IS[8] = 0x3D$$

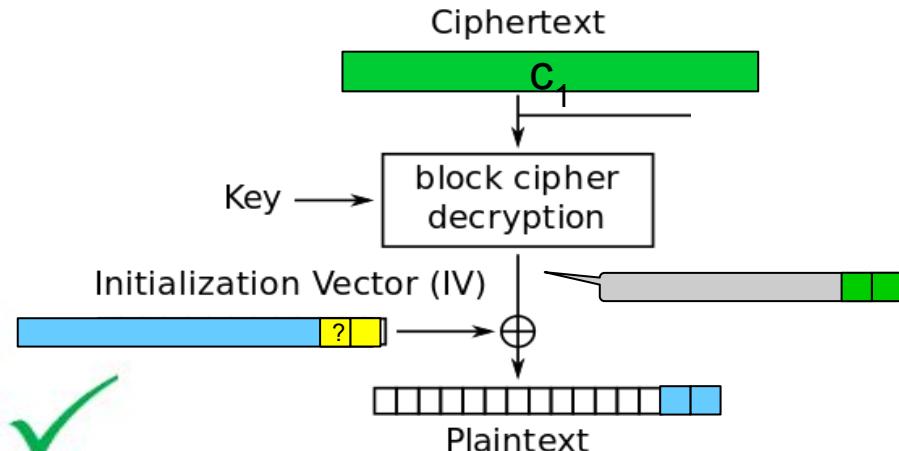


CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- Continue to the next byte

	Block 1 or 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x01
	VALID PADDING							



CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- To continue change last IV byte s.t. if produced a correct padding first

	1	2	3	4	5	6	7	8
Encrypted Input	0xE8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
Intermediate Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x00	0x3F						
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x02

TRIPLE DES

INVALID PADDING

The initialization vector byte 8, for the new padding, is calculated as follows:

$$IV[8] = IS[8] \oplus 0x02$$

$$IV[8] = 0x3D \oplus 0x02$$

$$IV[8] = 0x3F$$

Brute force search continues at next byte IS[7]



CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)
- Then brute force the byte 7 of the IV until a valid padding is reached

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x26	0x02	0x02
	VALID PADDING							

The initialization vector byte 8, for the new padding, is calculated as follows:

$$IV[8] = IS[8] \oplus 0x02$$

$$IV[8] = 0x3D \oplus 0x02$$

$$IV[8] = 0x3F$$

Brute force search continues at next byte IS[7]

$$IS[7] = 0x24 \oplus 0x02$$

$$IS[7] = 0x26$$

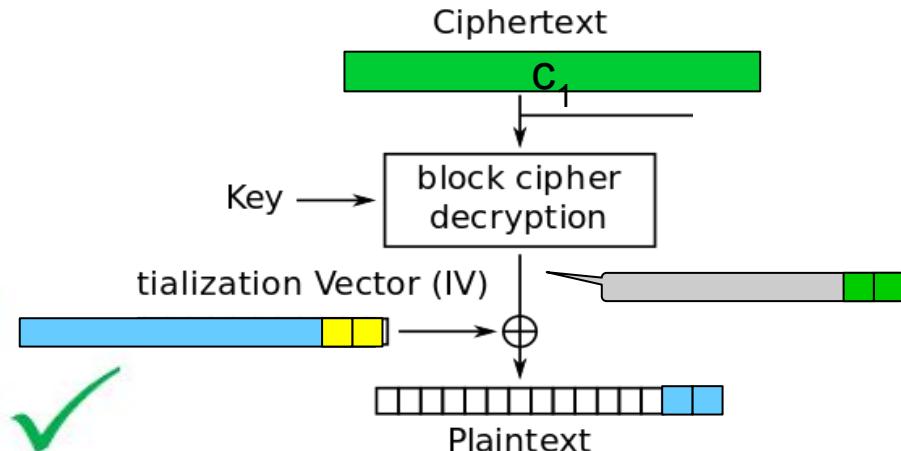


CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
	↓	↓	↓	↓	↓	↓	↓	↓
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x26	0x02	0x02
	VALID PADDING							



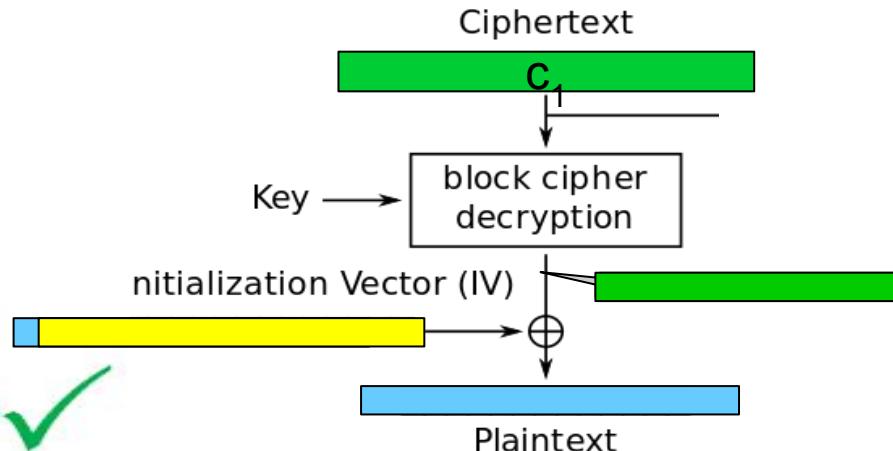
CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Reconstruct *intermediate state/value* (IS)

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x6B	0xFC	0x95	0x37
Intermediate Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x31	0x7B	0x2B	0x2A	0x0F	0x62	0x2E	0x35
Decrypted Value	0x08							

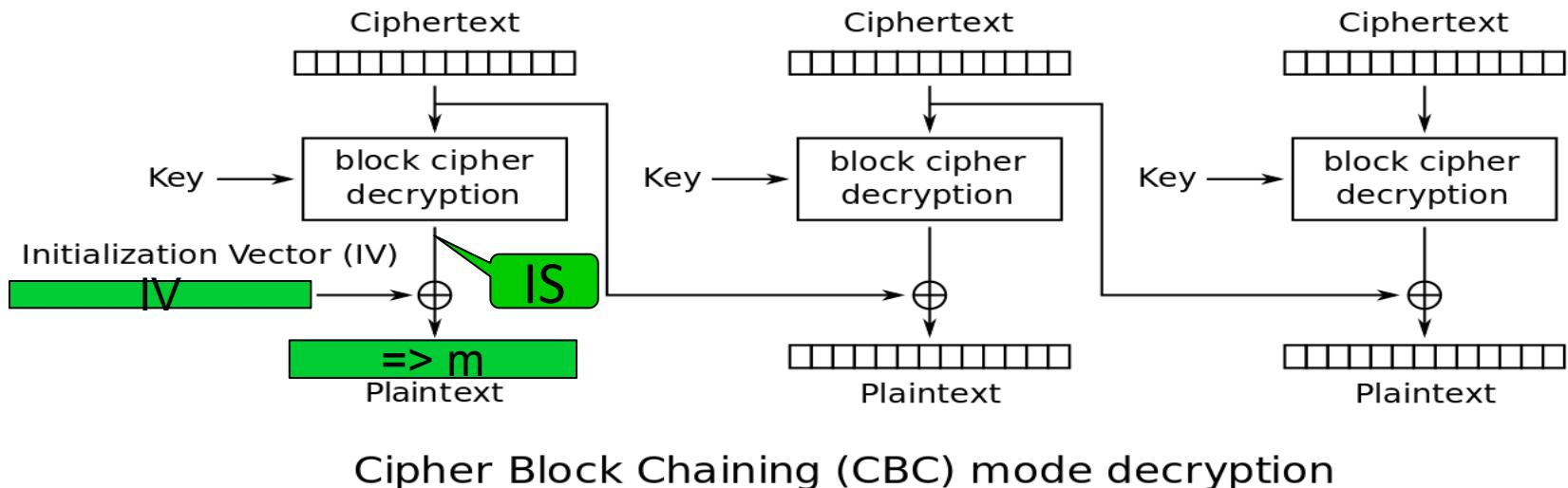
VALID PADDING



CBC

CBC Padding Oracle Attack (active; COA; side channel)

- Recover plaintext of original block using **original** IV and brute forced IS,
 $m = IV \oplus IS$



CTR



universität
wien

SBA
Research

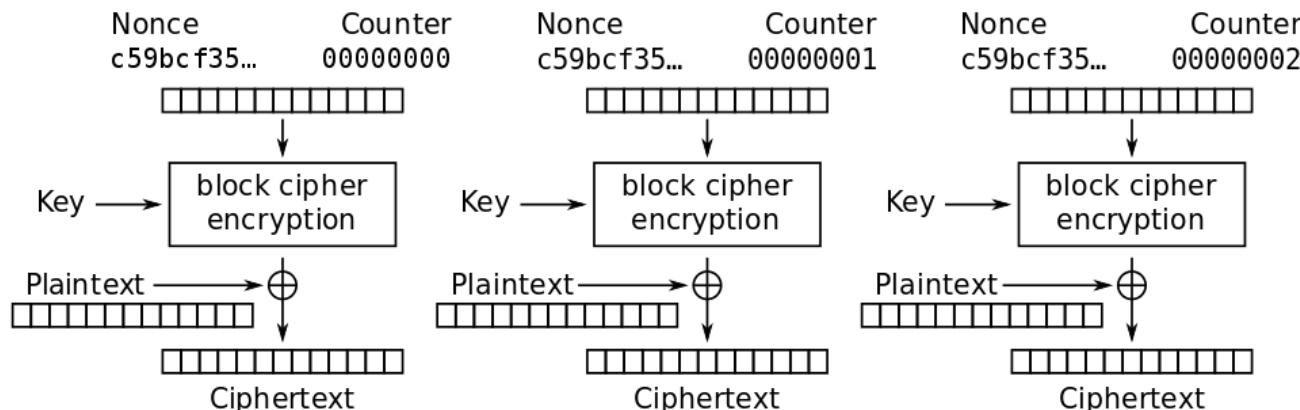
CTR

- Counter Mode (CTR)
- Pad message with random data so its length is a multiple of the block size
- Split message into blocks
- Encryption of one block depends on current counter value
- XOR every block with the output of the encryption function to produce the ciphertext



CTR

- Counter Mode (CTR)
 - encryption



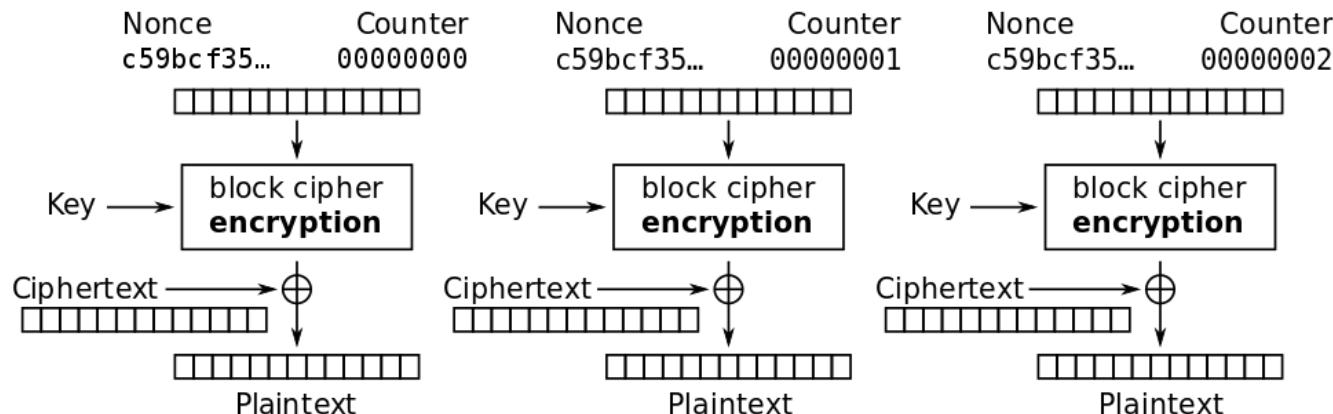
Counter (CTR) mode encryption

- [*] wikipedia.org



CTR

- Counter Mode (CTR)
 - decryption

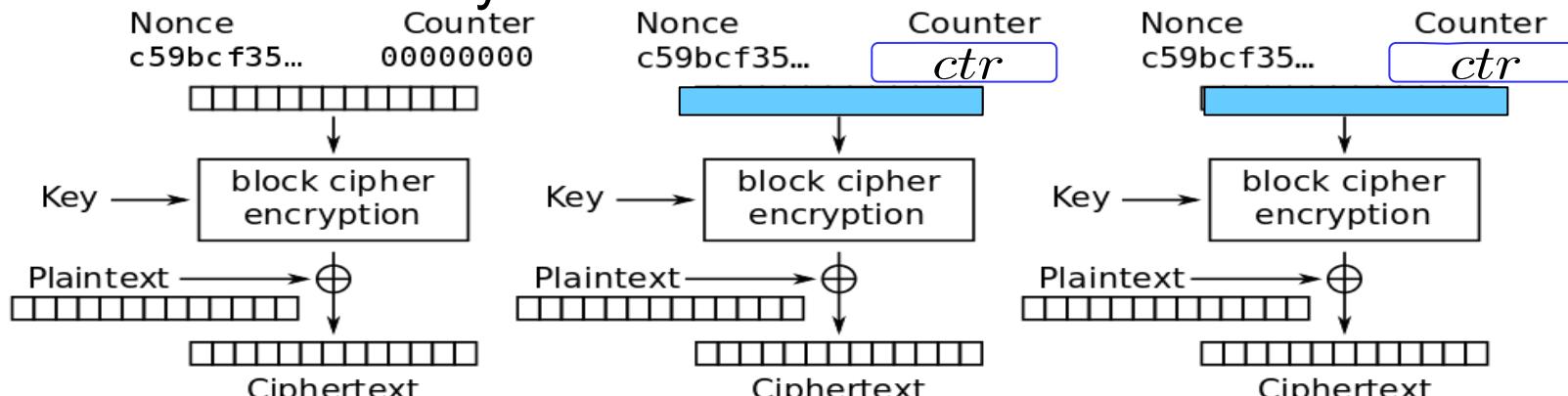


Counter (CTR) mode decryption

- [*] wikipedia.org

CTR

- What happens if counter is reused?
 - Duplicate ctr and nonce
 - over same key

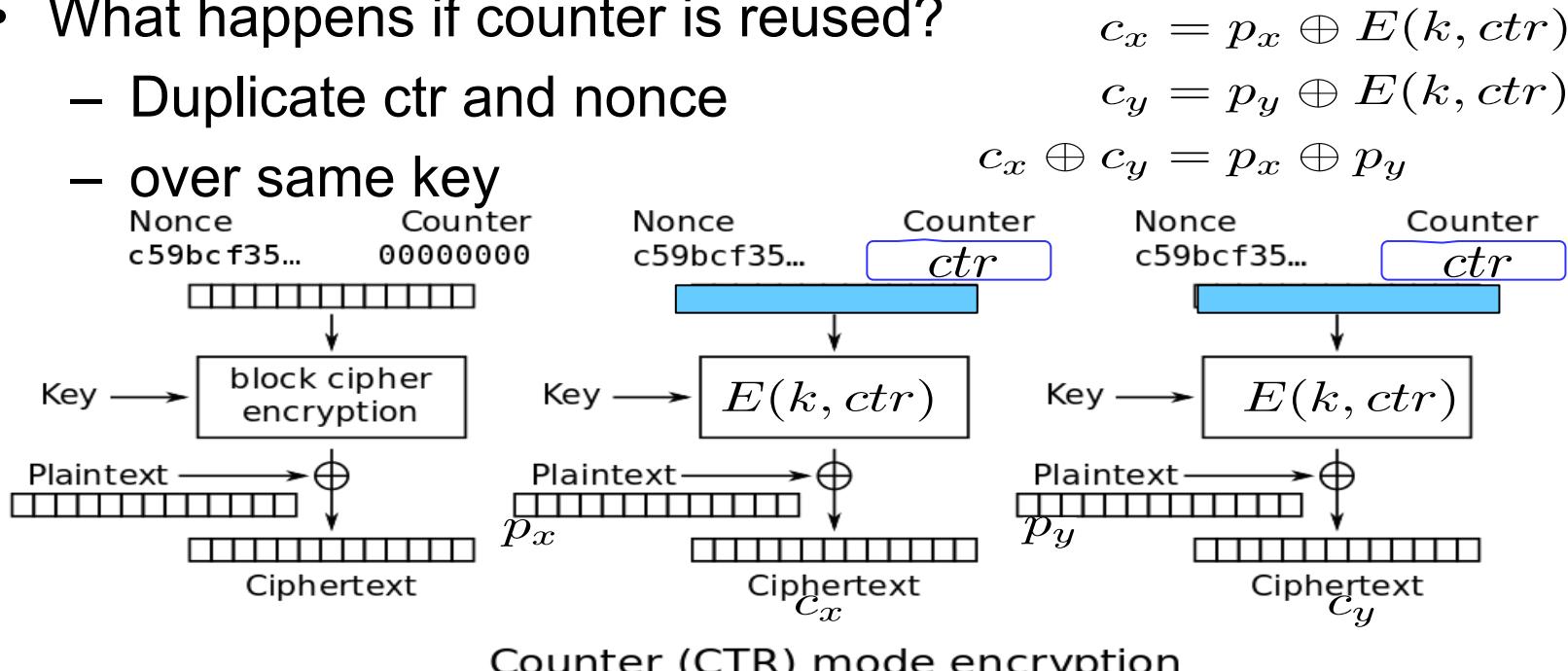


Counter (CTR) mode encryption



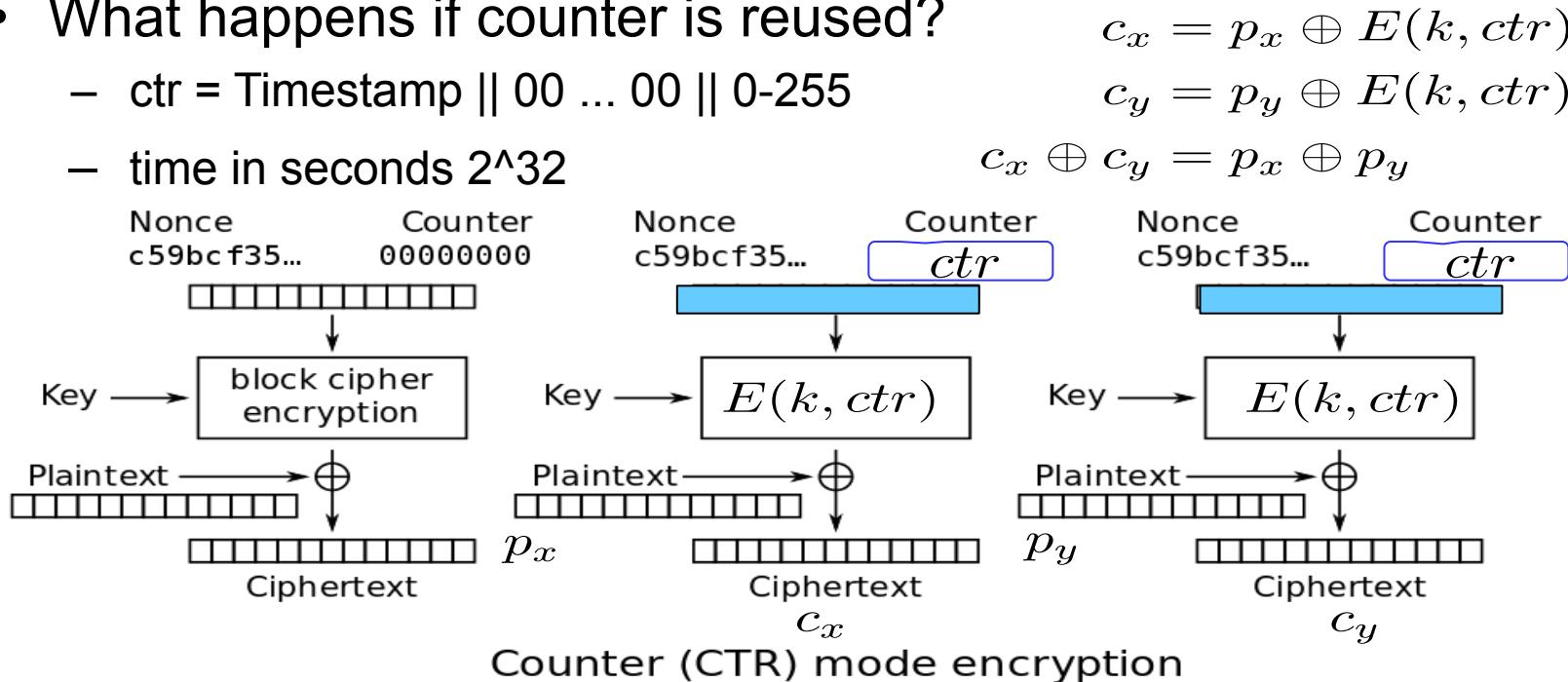
CTR

- What happens if counter is reused?
 - Duplicate ctr and nonce
 - over same key



CTR

- What happens if counter is reused?
 - $\text{ctr} = \text{Timestamp} \parallel 00 \dots 00 \parallel 0\text{-}255$
 - time in seconds 2^{32}



Cryptographic Hash Functions



Hash Function

- A **hash function** (H) takes a **message** (m) of arbitrary but finite size and outputs a fixed size **hash** (a.k.a. *digest*) (h)

message

“Hello this is some example test message which is longer than the output hash value”

$H(\text{message})$

hash

c7bd6bef1fa3a7be2d47616931573298dcfb84ee

Cryptographic Hash Function

- . There are 4 additional properties of a *hash function* to qualify for a ***cryptographic hash function***:
- 1)
 - 2)
 - 3)
 - 4)



Cryptographic Hash Function

- . There are 4 additional properties of a *hash function* to qualify for a ***cryptographic hash function***:
- 1) Easy to compute
 - 2) Pre-image resistance
 - 3) Second pre-image resistance
 - 4) Collision resistance



Cryptographic Hash Function

(1) easy to compute the hash of any given message

$$h = H(m), \text{ where } h \text{ is of fixed length}$$

message

“Hello this is some example test message which is longer than the output hash value”

$H(\text{message})$

this is easy

hash

c7bd6bef1fa3a7be2d47616931573298dcfb84ee

Cryptographic Hash Function

(2) Pre-image resistance:

infeasible* to generate a message that has a given hash (therefore also called *one-way-function*)

Given a hash h it should be infeasible to find any message m s.t. $h = H(m)$

Message (pre-image)
?

this is not possible
only through
brute force search

hash

c7bd6bef1fa3a7be2d47616931573298dcfb84ee

Cryptographic Hash Function

(2) Pre-image resistance:

infeasible* to generate a message that has a given hash (therefore also called *one-way-function*)

Question: Does this infeasibility depend on something else
Besides the hash function?

Message (pre-image)
?

this is not possible
only through
brute force search

hash

c7bd6bef1fa3a7be2d47616931573298dcfb84ee

Cryptographic Hash Function

(2) Pre-image resistance:

e.g. m is the result of a coin-flip then we can deduce m from in one try $H(m)$

So a hash function does not automatically hide its input if the input is guessable!

Hiding. A hash function H is hiding if: when a secret value r is chosen from a probability distribution that has *high min-entropy*, then given $H(r||m)$ it is infeasible to find m .

Cryptographic Hash Function

(2) Pre-image resistance

- . Real world example of misuse:
 - Use hash function to obfuscate stored IPv4 addresses for privacy reasons
 - $2^{32} \approx 4$ billion addresses
 - . Without throwing away ~350 million for private ranges
 - 10-25 million hashes per second for a desktop PC is reasonable depending on the hash function
 - Reverse lookup table (rainbow table) calculated within seconds
 - Hashing does not magically increase entropy!

Cryptographic Hash Function

(3) Second pre-image resistance:

infeasible to modify a message without changing the resulting hash

Given a message m it should be infeasable to find another message m' such that $m \neq m'$ and $H(m) = H(m')$

message

"Hello this is some example test message which is longer than the output hash value"

$H(\text{message})$

hash

c7bd6bef1fa3a7be2d47616931573298dcfb84ee

Cryptographic Hash Function

(3) Second pre-image resistance:

infeasible to modify a message without changing the resulting hash

Given a message m it should be infeasable to find another message m' such that $m \neq m'$ and $H(m) = H(m')$

message

“hello this is some example test message which is longer than the output hash value”

$H(\text{message})$

small change in input
large change in hash

hash

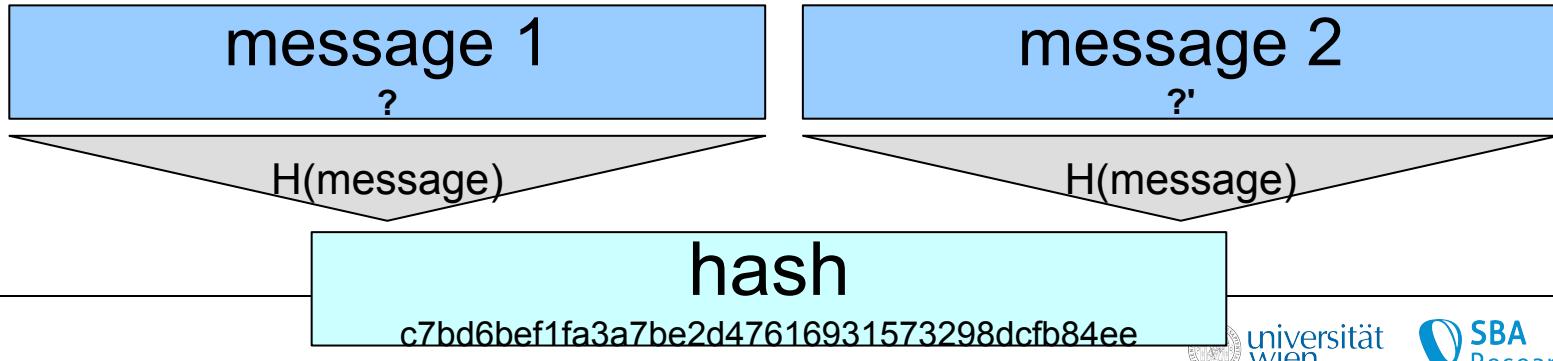
08946ff1a58ef31d5a0f4cdd455d66a18cb4c01c

Cryptographic Hash Function

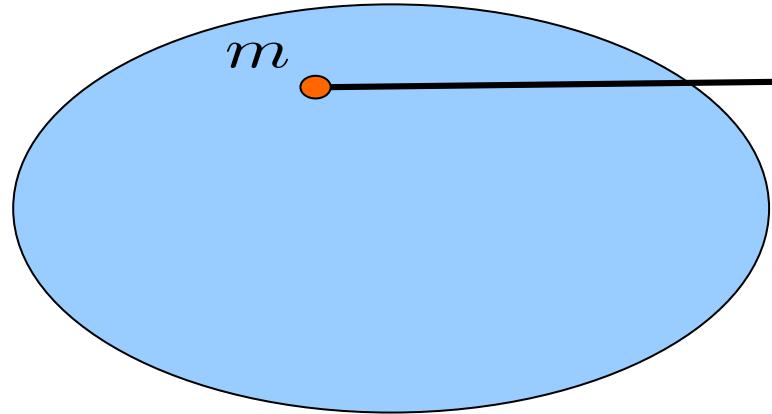
(4) Collision resistance:

infeasible to find **any** two different messages with the same hash

It should be infeasable to find **any** two messages m and m' where $m \neq m'$ and $H(m) = H(m')$

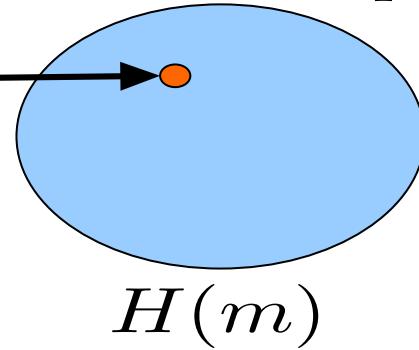


512 input bits



H

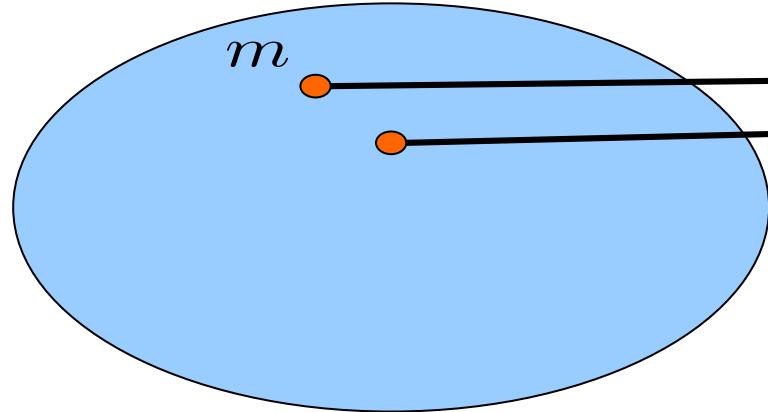
256 output bits



$H(m)$

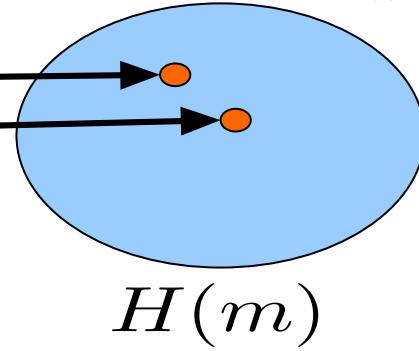


512 input bits



H

256 output bits

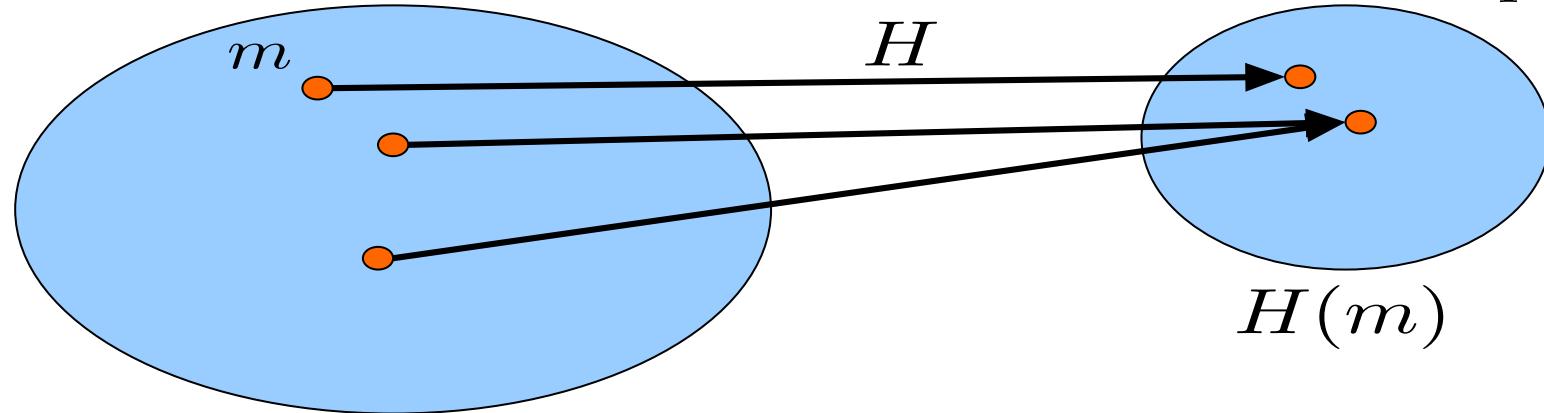


$H(m)$



Collision

512 input bits



256 output bits

$H(m)$

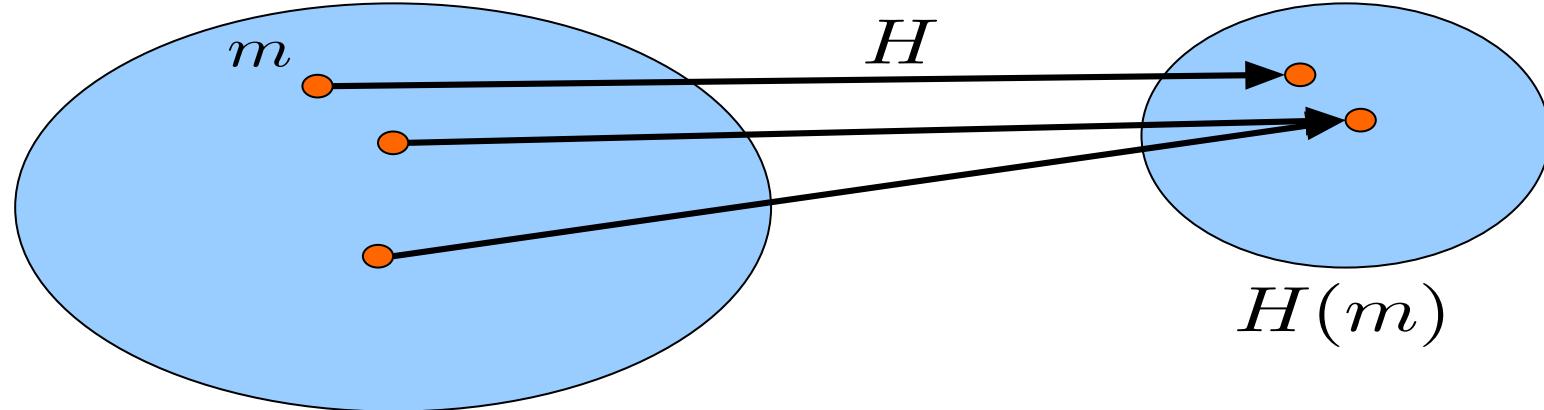


universität
wien

SBA
Research

Collision

512 input bits

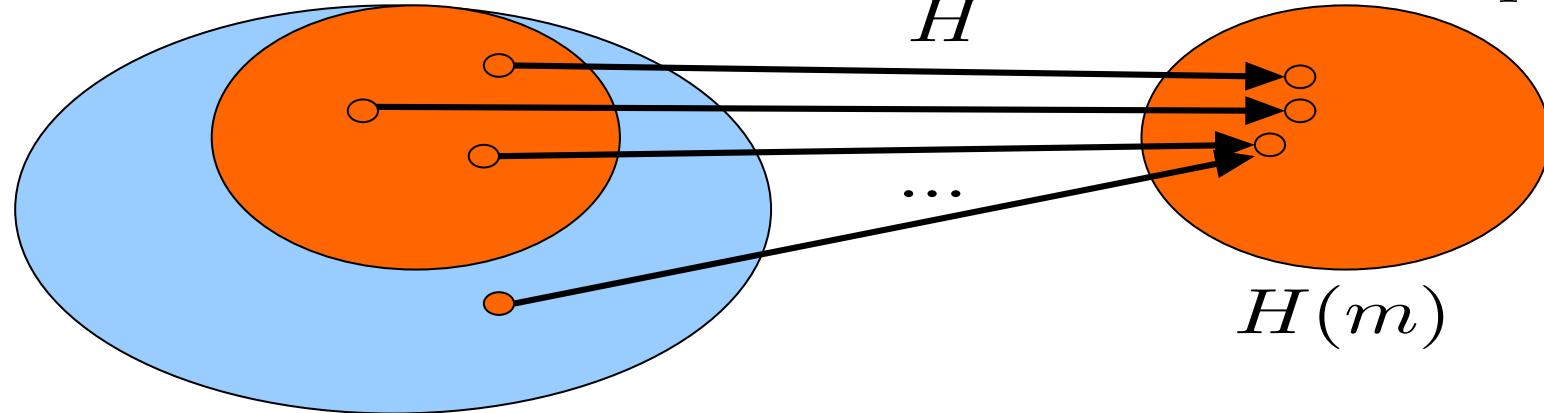


256 output bits

What is the maximum number of guesses required
to certainly find a collision? And what is the space complexity?

Collisions

512 input bits

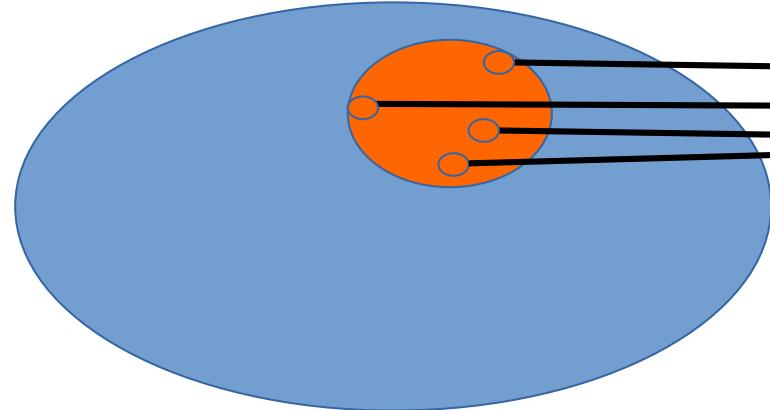


256 output bits

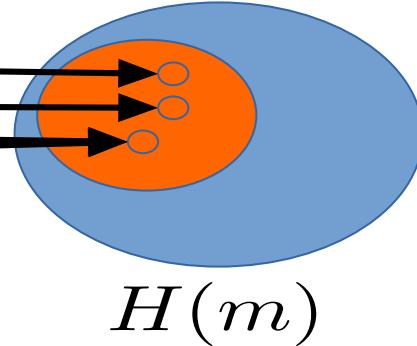
- The maximum number of guesses required to **certainly** find a collision is $2^{256} + 1$ $O(2^n)$ time complexity $O(2^n)$ space complexity, where $n = \text{len}(H)$

Collisions

512 input bits



256 output bits



- **Birthday bound:** ~50% probability of a collision after $\approx 2^{128}$
 $O(2^{n/2})$ time complexity
 $O(2^{n/2})$ space complexity,
where $n = \text{len}(H)$

Collisions

It takes approx. 10^{27} years to calculate 2^{128} hashes

“if every computer ever made by humanity was computing since the beginning of the entire universe, up to now, the odds that they would have found a collision is still infinitesimally small. So small that it’s way less than the odds that the Earth will be destroyed by a giant meteor in the next two seconds.” [1]

-
- [1] [narayanan2016bitcoin]

Size Comparison

- It takes approx. 10^{27} years to calculate 2^{128} hashes
- There are approx. $7 \cdot 10^{27}$ atoms in a human body [1]
- There are approx. 10^{82} atoms in the observable universe [1]
- Let's compare this to a 256 bit hash value:

```
>>> import math  
  
>>> math.log10(2**256)  
77.063d67888997919  
  
>>> math.log2(10**82)  
272.39810378076373
```

A 256 bit hash has approx. 10^{77} possible outputs

A 273 bit hash would have more possible output values than there are atoms in the observable universe

[1] <https://www.livescience.com/how-many-atoms-in-universe.html>

Usage of Hash Functions

- Proof-of-Work
- Hash chains (also before blockchain e.g., git)
- Merkle trees
- Store passwords
- Bloom filter
- MAC (Message authentication code)
- Used in context of digital signatures
- ...



Proof-of-Work (PoW)

PoW first proposed by Dwork and Naor [dwork1992pricing]

What we want of a PoW scheme:

- Easy to verify
- Difficult to create
- Parameterizable difficulty



Hash based PoW

Hash/ Search puzzle [1]

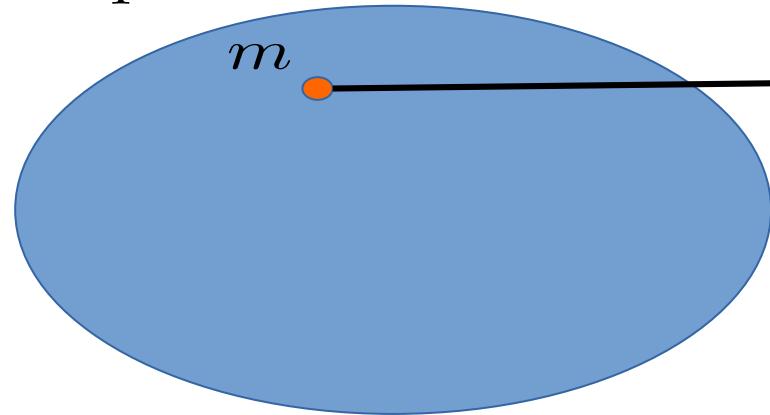
- Based on **partial pre-image attack**
 - Basically we want the resulting hash of an input to have certain properties (i.e. start with a certain amount of 0 bits)

The puzzle consists of:

- The input data x
- A random value r
- A target set S
- A solution is a input value such that: $m = r||x \quad H(m) \in S$
- **In simple terms:**
try different values for r , concatenate them with x and apply the hash function H until the resulting hash is part of the target set S

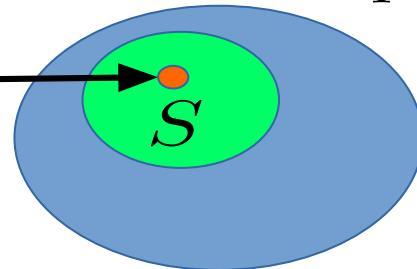
Hash based PoW

512 input bits



H

256 output bits



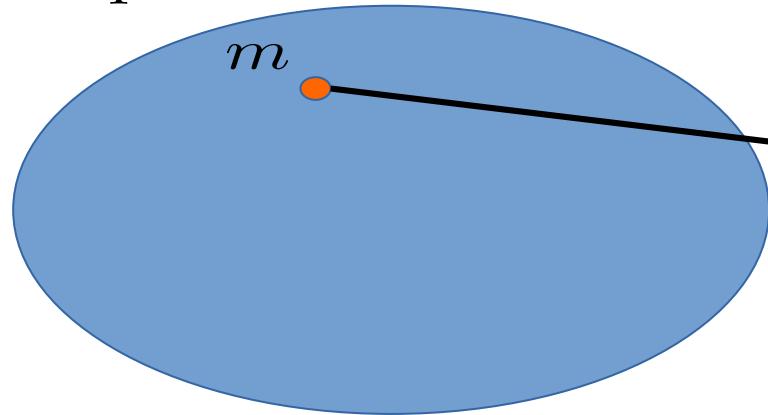
$$H(m) \in S$$

- m is a valid puzzle solution



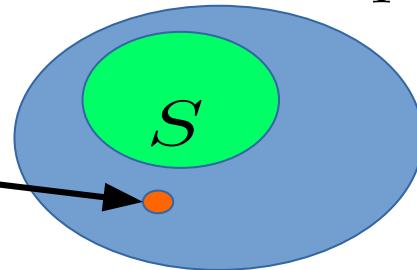
Hash based PoW

512 input bits



H

256 output bits

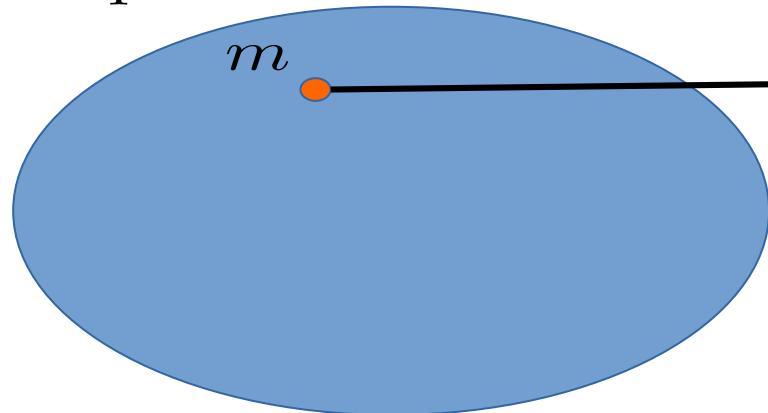


$$H(m) \notin S$$

- m is no valid puzzle solution

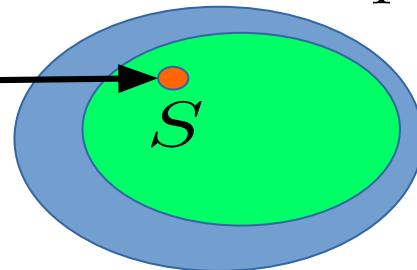
Hash based PoW

512 input bits



H

256 output bits



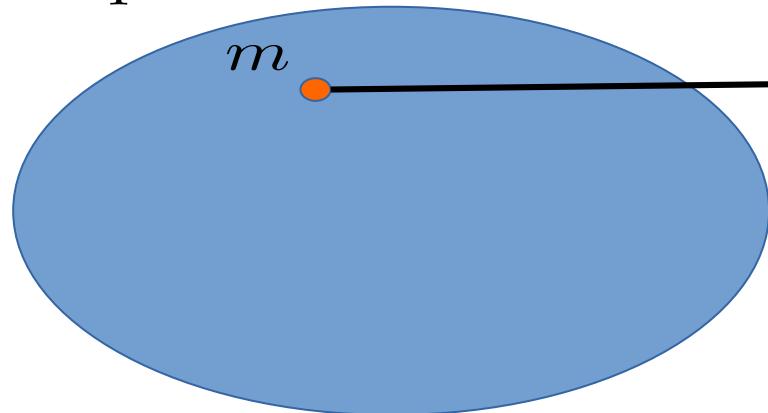
$$H(m) \in S$$

- Size of S determines the difficulty
- If S is large, PoW is less difficult
- In Bitcoin S is defined by the number of leading zeros of SHA-256



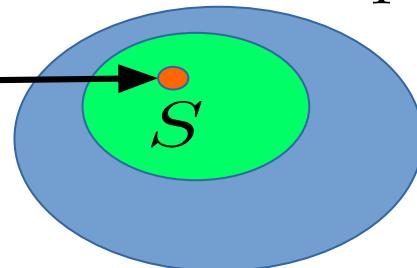
Hash based PoW

512 input bits



H

256 output bits



$$H(m) \in S$$

- Size of S determines the difficulty
- If S is large, PoW is less difficult
- In Bitcoin S is defined by the number of leading zeros of SHA-256



Hash based PoW

- **Check condition:**

- Find any hash of len. (n) that starts with certain amount of zero bits (t)

$$H(m) < 2^{(n-t)}$$

or

$$H(m)/2^{(n-t)} = 0$$

```
>>> n = 4  
>>> t = 2  
>>> 0b0111 < 2**(n-t)  
False  
>>> 0b0111 // 2**(n-t)  
1
```

```
>>> 0b0010 < 2**(n-t)  
True  
>>> 0b0011 < 2**(n-t)  
True  
>>> 0b0011 // 2**(n-t)  
0
```

- Proof-of-Work - First Application
- Hashcash: connect proof-of-work to application

- *version* and **target bits** (number of zero bits required)
- **date**
- **resource** string, e-mail address
- **random** characters
- **nonce**, *counter* incremented for brute force search

$$m = ver || bits || date || resource || rand || nonce$$
$$H(m) < 2^{(n-t)}$$

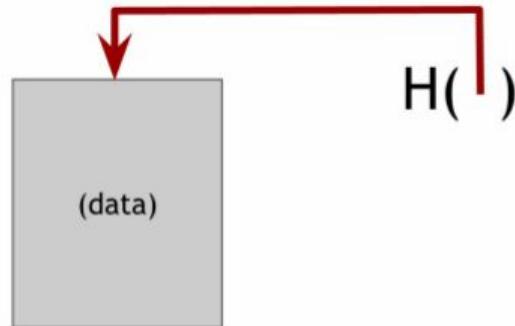
```
$ echo -n "1:20:1303030600:adam@cypherspace.org::McMybZlhxKXu57jd:ckvi" | sha1sum  
00000b7c65ac70650eb8d4f034e86d7d5cd1852f
```

- [1] <http://en.wikipedia.org/wiki/Hashcash>



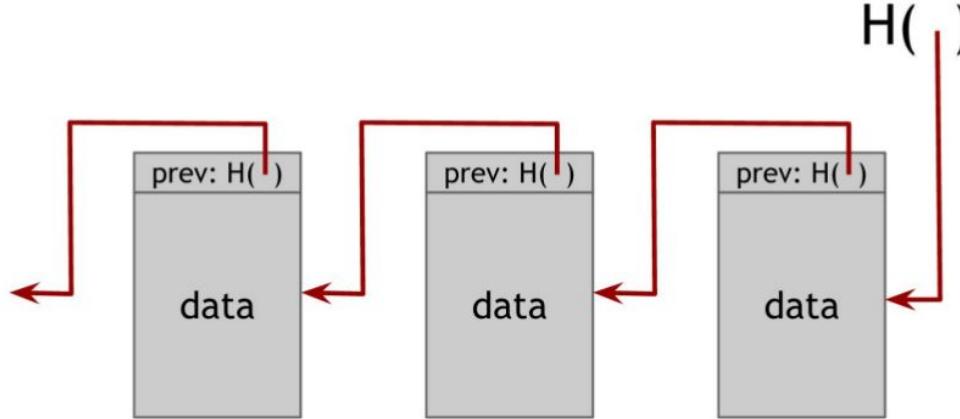
Hash pointers

- “A hash pointer is a **pointer to where data is stored** together with a **cryptographic hash of the value** of that data at some fixed point in time.” [1]



• [1] [narayanan2016bitcoin]

Hash chains: Chained Hash Pointers aka. “Blockchain”



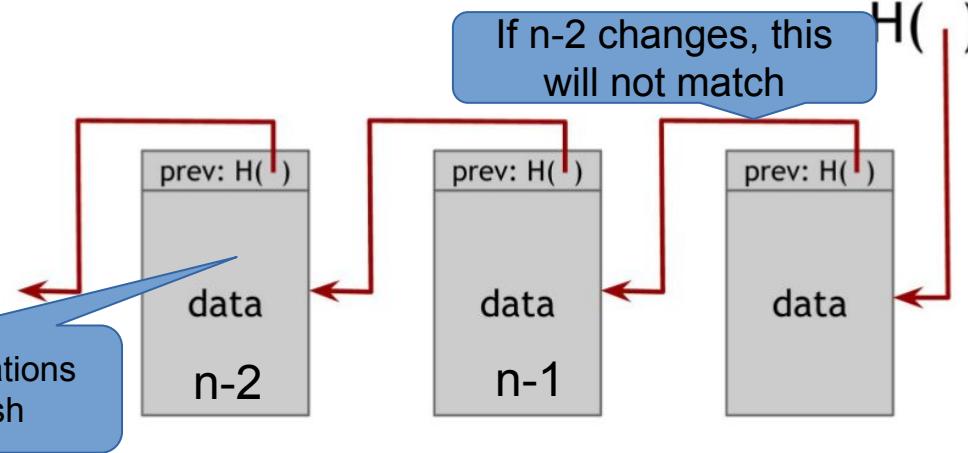
*“A block chain is a **linked list** that is built with **hash pointers** instead of pointers” [1]*

Blocks contain the digest (hash) of their predecessor, so one can verify that **previous elements have not changed**

[1] [narayanan2016bitcoin]

Hash chains:

Chained Hash Pointers aka. “Blockchain”



*“A block chain is a **linked list** that is built with **hash pointers** instead of pointers” [1]*

Blocks contain the digest (hash) of their predecessor, so one can verify that **previous elements have not changed**

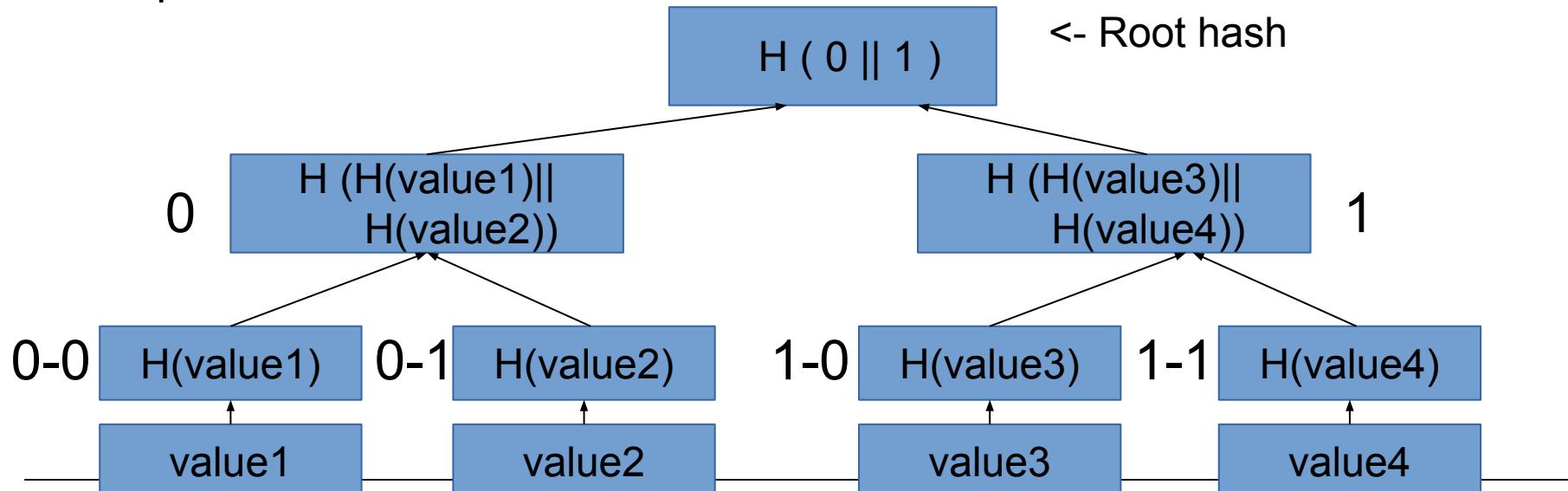
- [1] [narayanan2016bitcoin]

Merkle Trees



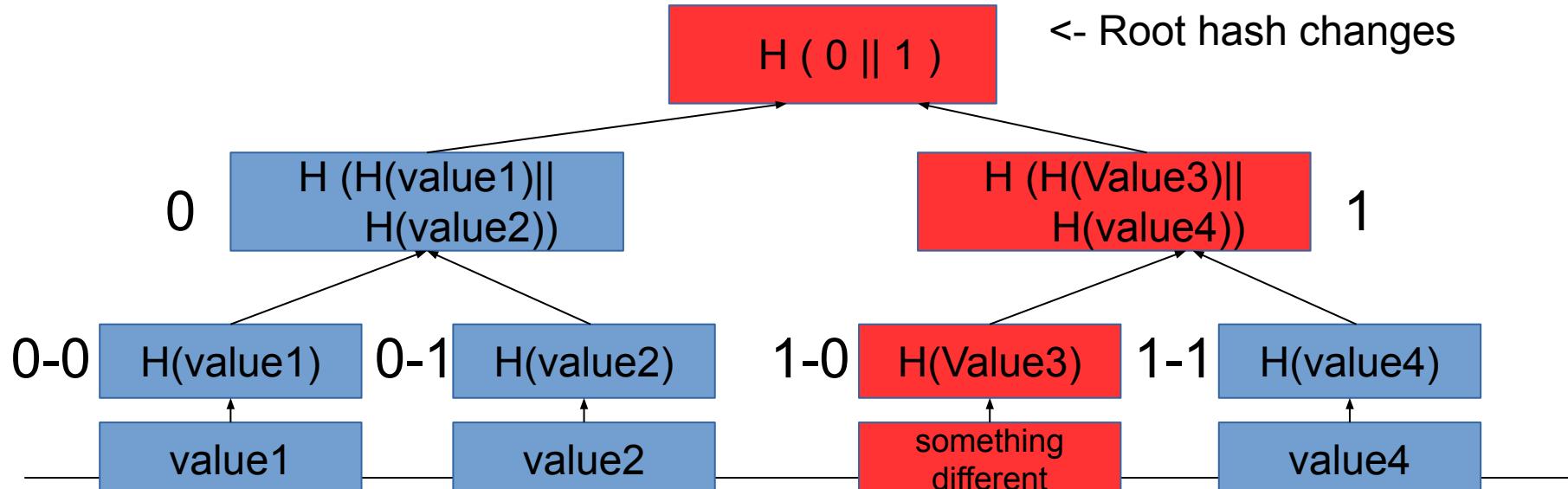
Merkle tree

- **Data blocks**, are **hashed and grouped in pairs**.
- The hash of each of these pairs is stored in a parent node all the way up the tree until we reach the root node



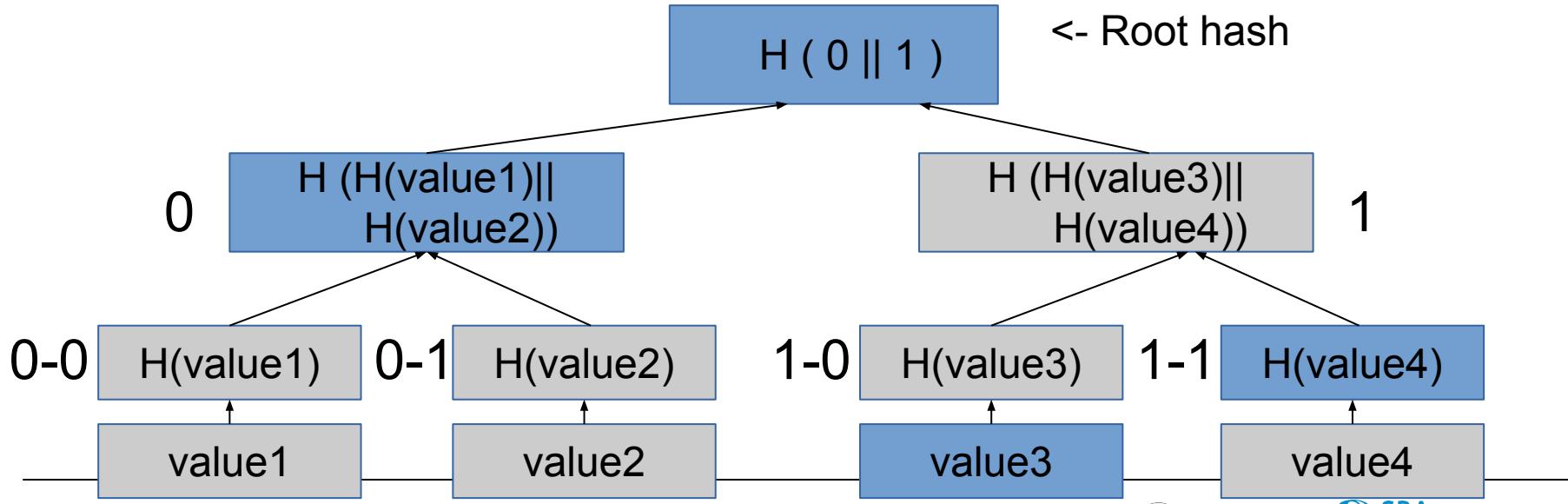
Merkle tree change propagation

- Also used in other scenarios when compact commitments are needed (e.g., Cryptocurrencies, Timestamping, ...)
- If one leaf changes, change is propagated to root hash:

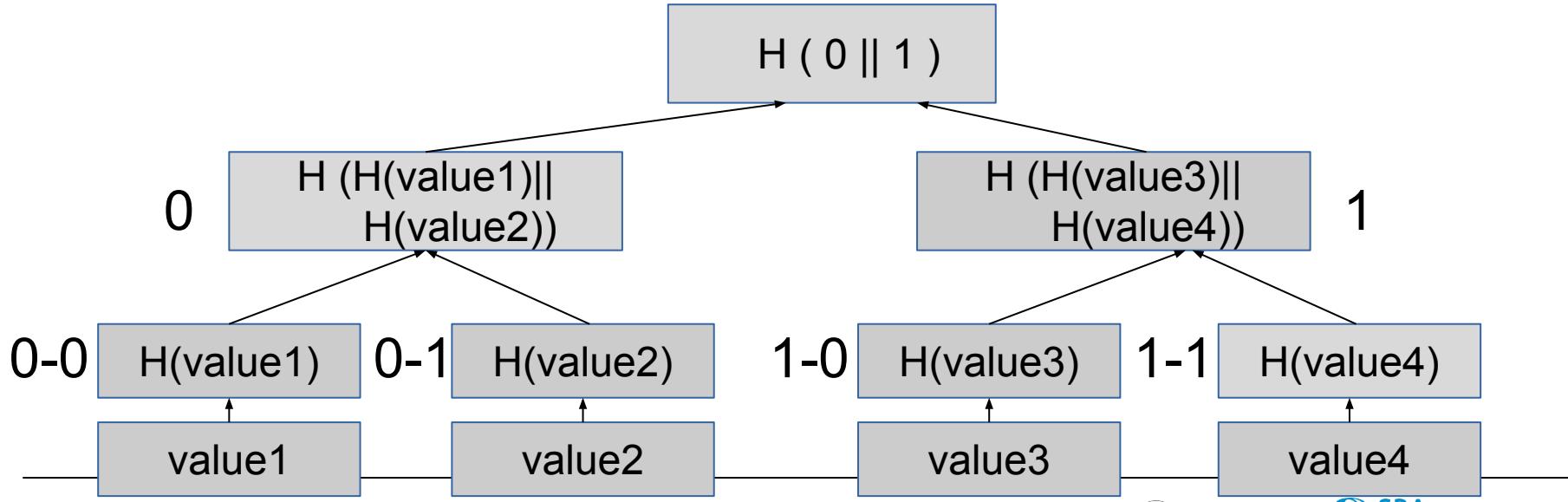


Merkle tree inclusion proof

- If there are N values to be stored in leafes, tree height (number levels) is $\log_2(N)$
- The number of elements to provide is therefore $\log_2(N)$, so verification can be done in $O(\log_2(N))$ steps.

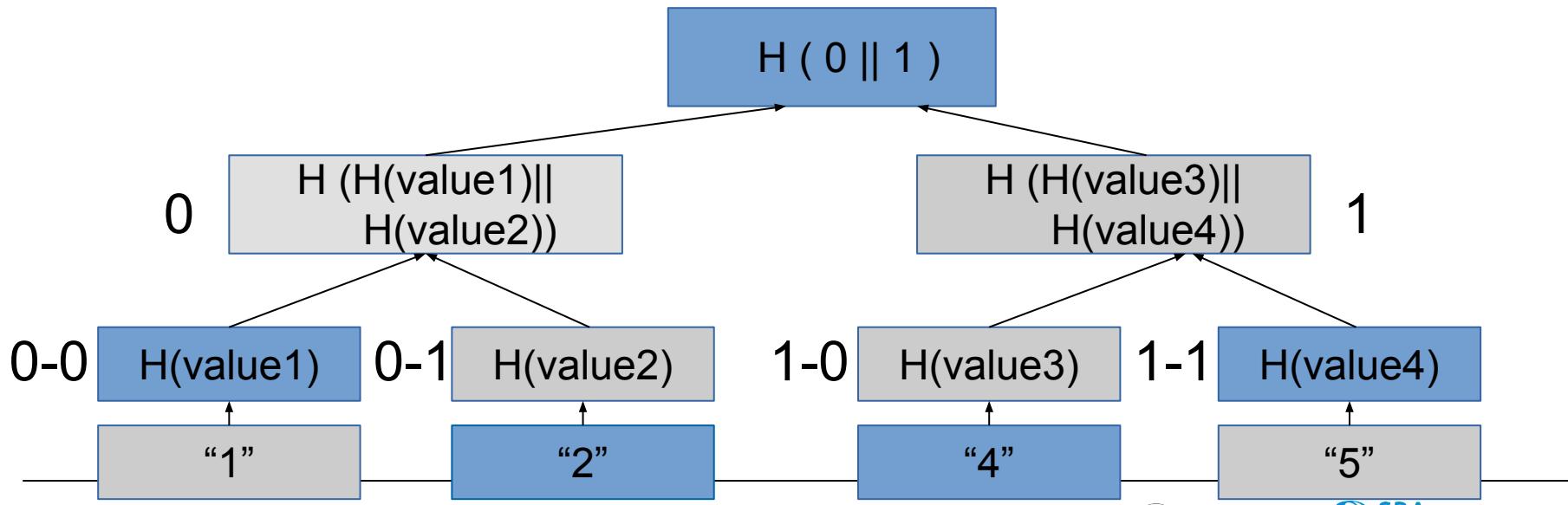


Non-inclusion proof?



Non-inclusion proof?

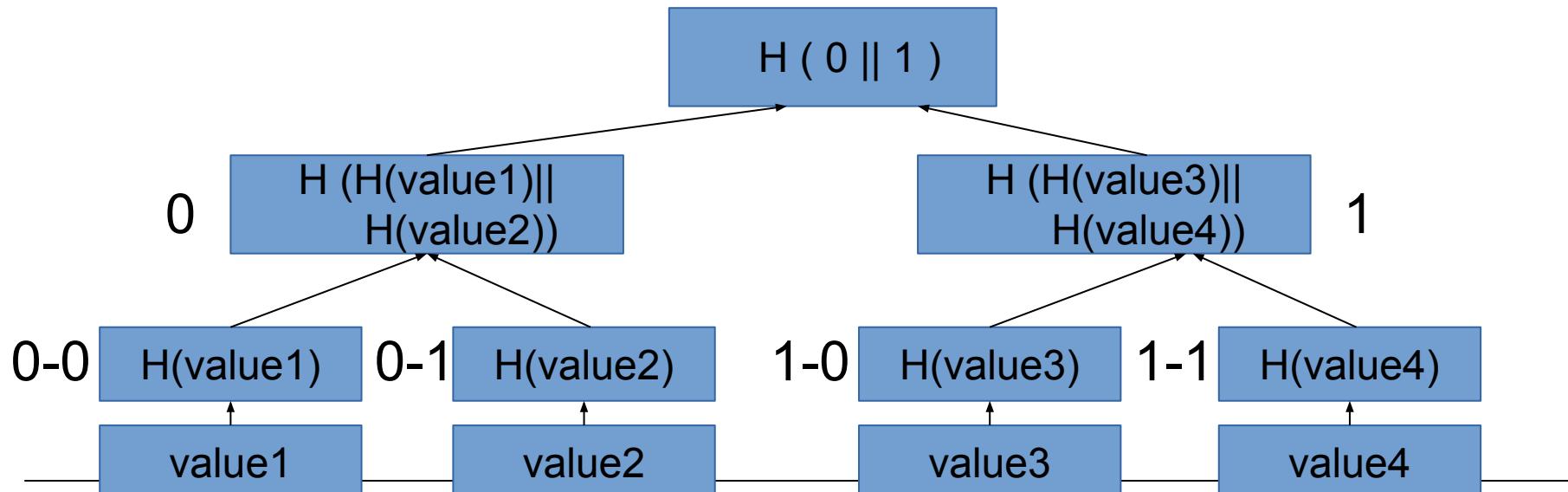
- Generally **not** possible, only if values are guaranteed to be ordered!



[1] <https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf>

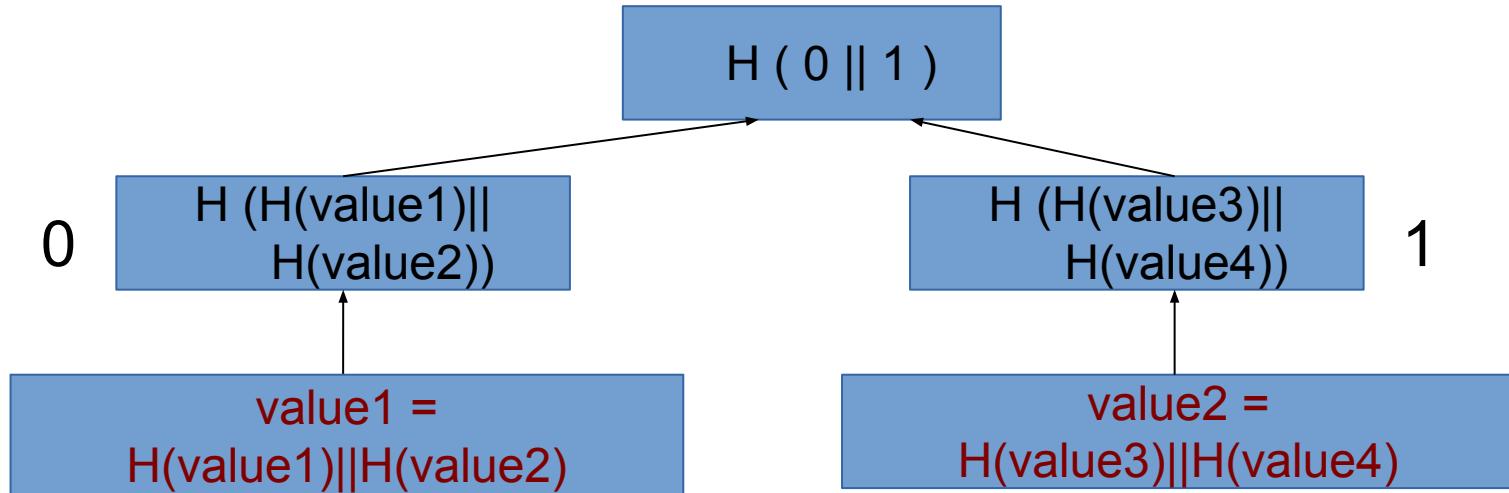
Merkle tree

- Under which conditions is a second pre-image attack possible?
(i.e., provide a proof for a value that was not in the original list of values that generated the root hash)



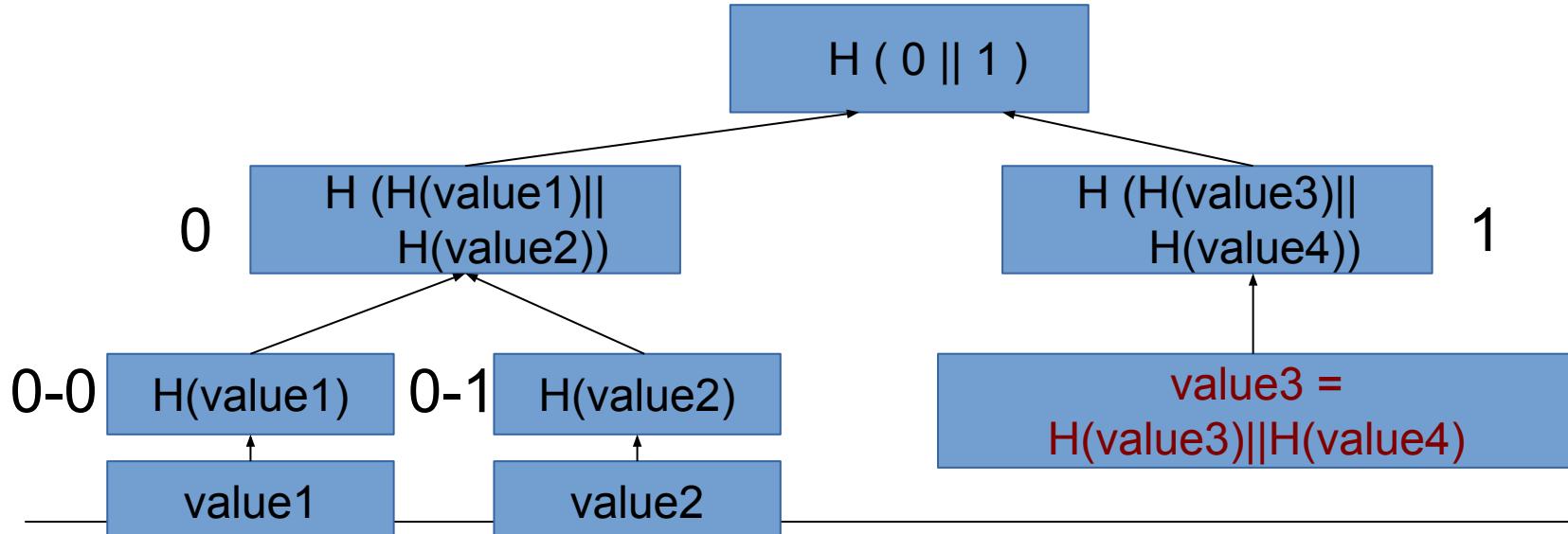
Second preimage attack

- The last layer is striped, but it results in the same root hash hashed and grouped in pairs.



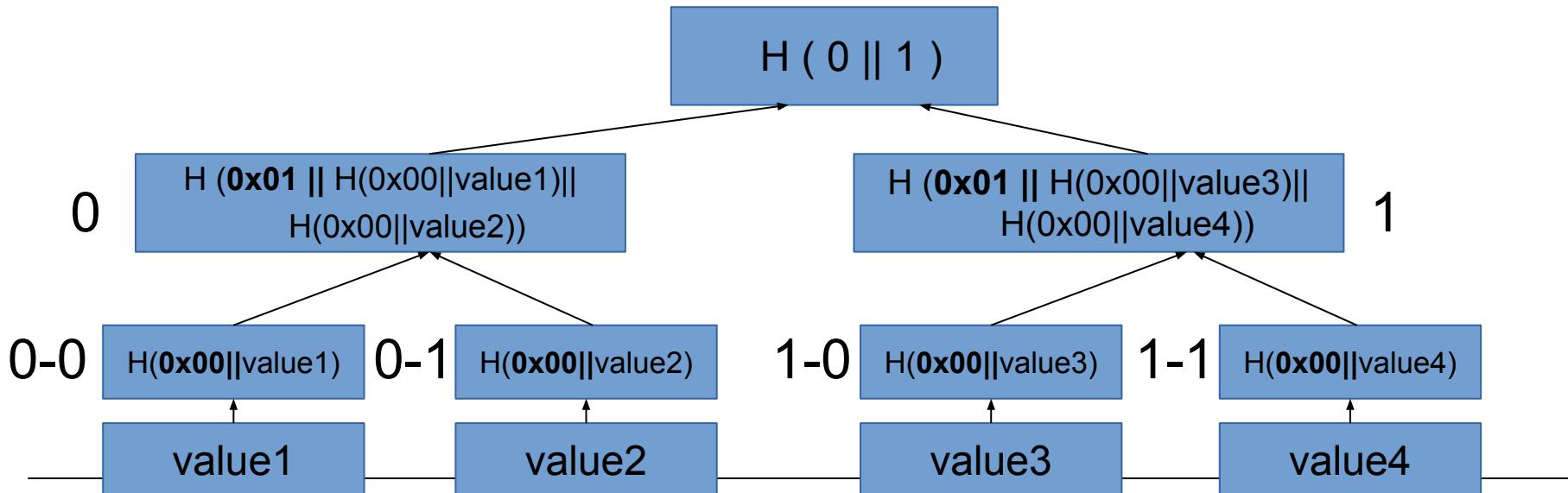
Second preimage attack

- If unbalanced trees are allowed this is also possible.



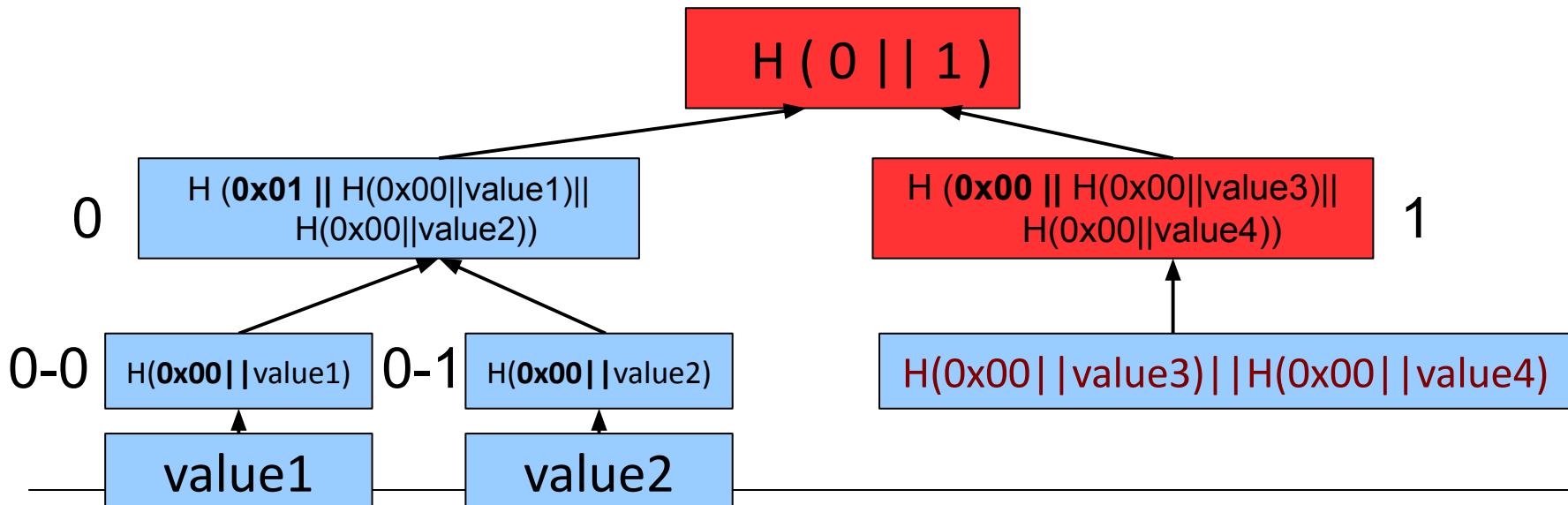
Fix second preimage attack

- Indicate or fixate the depth of the tree, or
- Prepend **0x00** to leaf nodes and **0x01** to other nodes, or
- Use different hash function for first step (leafs)



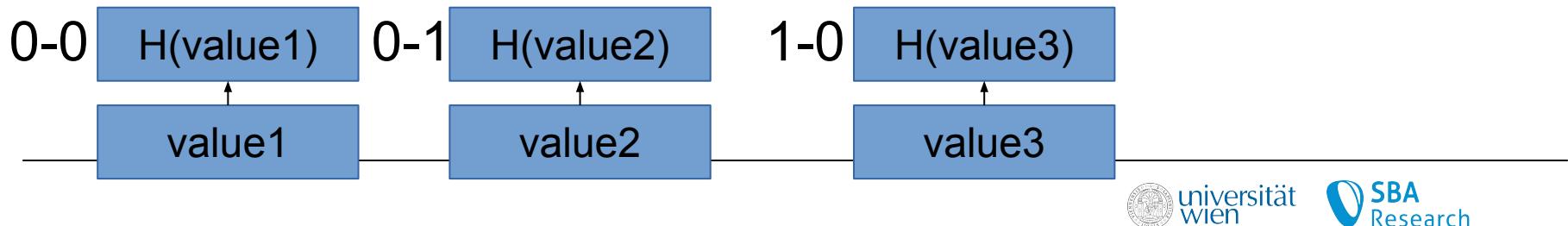
Fix second preimage attack

- Indicate or fixate the depth of the tree, or
- Prepend **0x00** to leaf nodes and **0x01** to other nodes, or
- Use different hash function for first step (leafs)



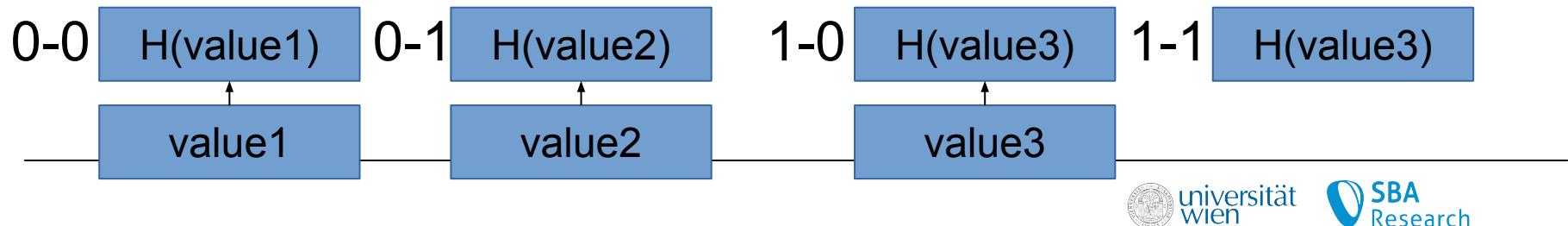
Merkle trees in Bitcoin

- How to not handle an uneven number of values.



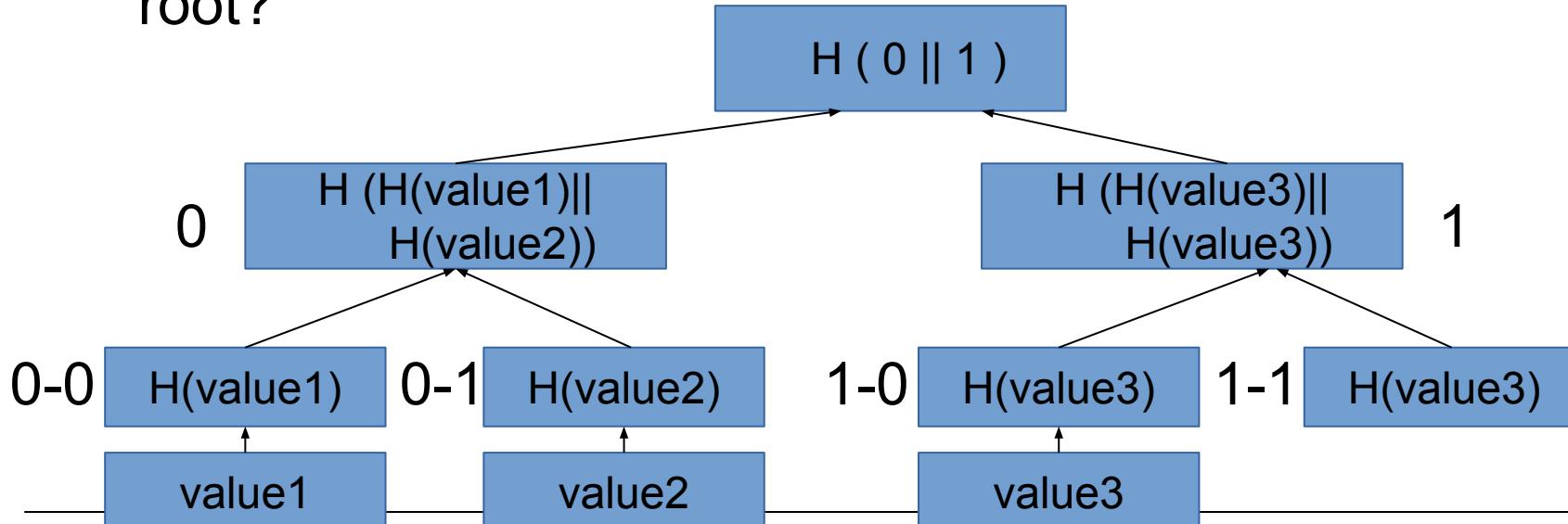
Merkle trees in Bitcoin

- In Bitcoin the last inner node is duplicated:



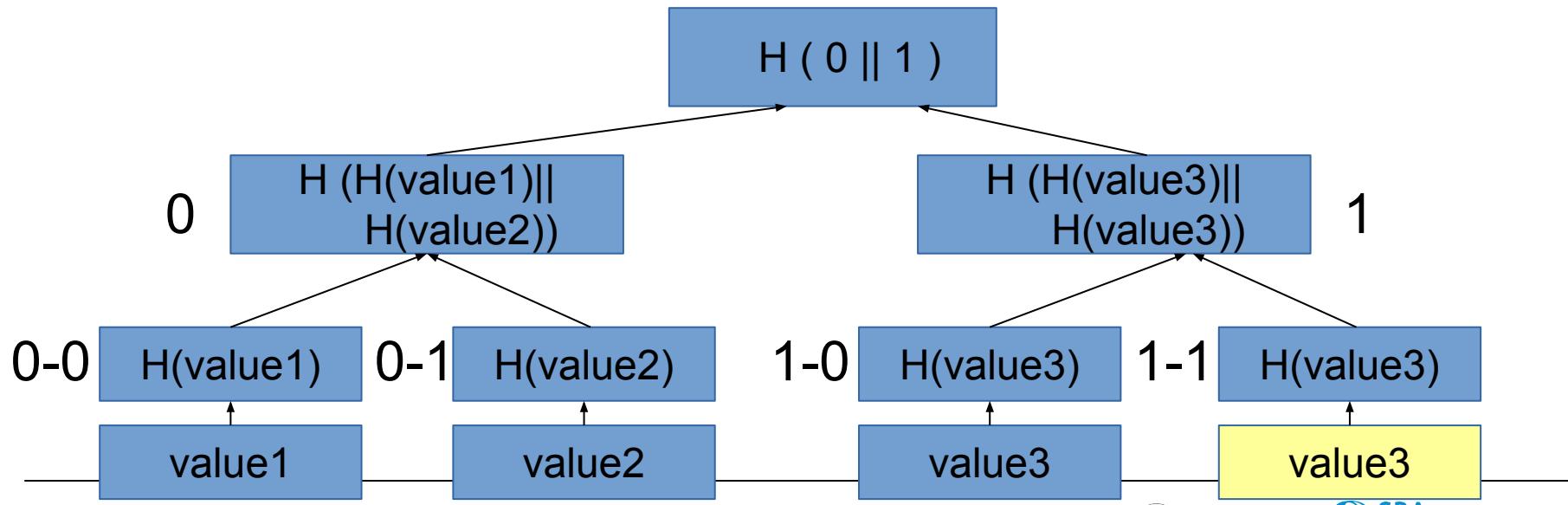
Merkle trees in Bitcoin

- In Bitcoin the last inner node is duplicated.
- Can you create different data which maps to the same root?



Merkle trees in Bitcoin

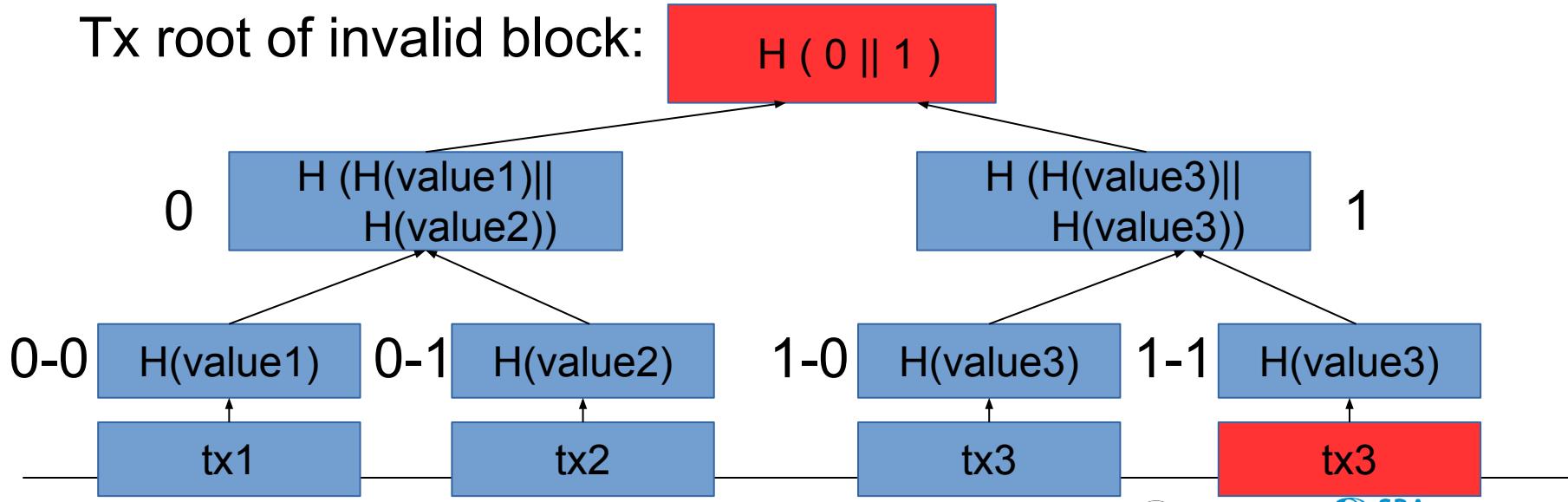
- Yes, just add the last leaf twice:



Merkle trees in Bitcoin

- In Bitcoin this resulted in CVE-2012-2459, DoS attack on Bitcoin network. Once a block header (hence a root hash) is marked invalid, any further attempts to submit block are rejected => Networksplit

Tx root of invalid block:

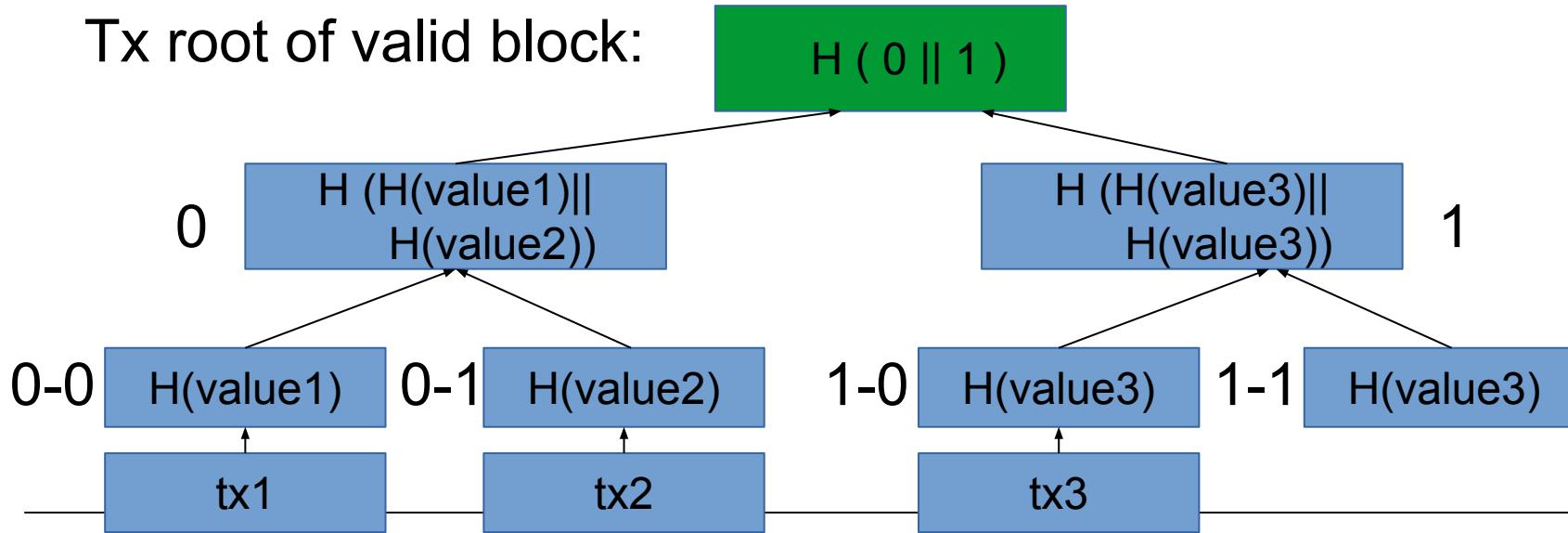


[1] <https://github.com/bitcoin/bitcoin/blob/master/src/consensus/merkle.cpp>

Merkle trees in Bitcoin

- In Bitcoin this resulted in CVE-2012-2459, DoS attack on Bitcoin network. Once a block header (hence a root hash) is marked invalid, any further attempts to submit block are rejected => Networksplit

Tx root of valid block:

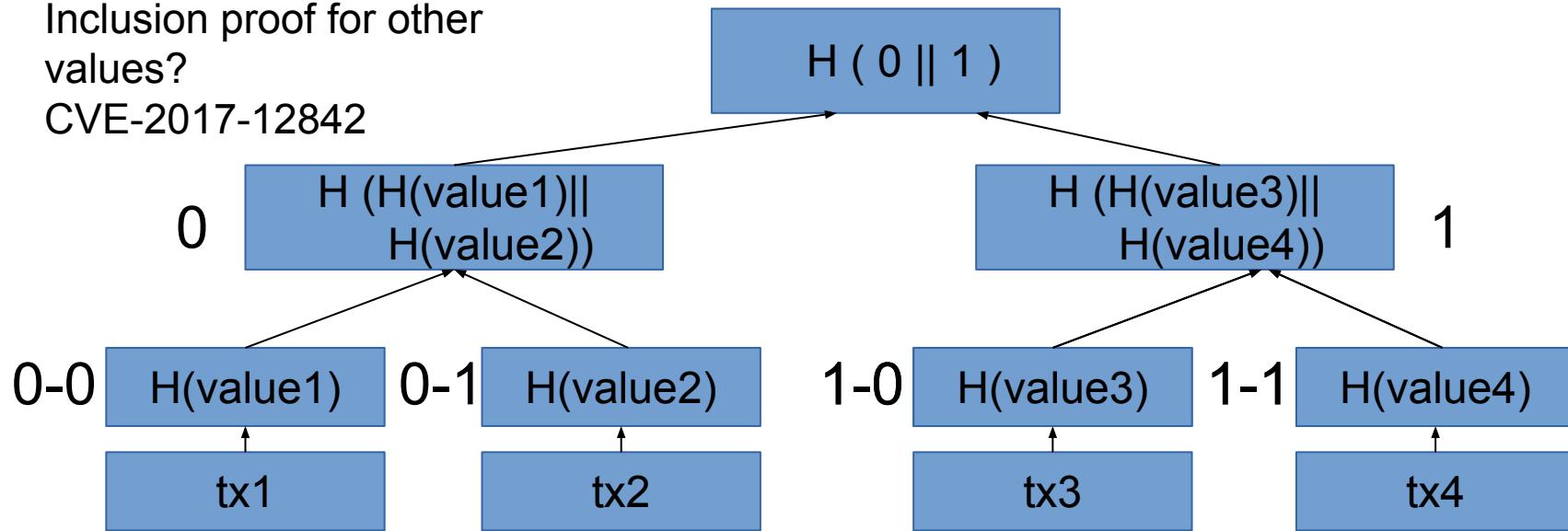


- [1] <https://github.com/bitcoin/bitcoin/commit/be8651dde7b59e50e8c443da71c706667803d06d>

Merkle trees in Bitcoin

- Inclusion proof for other values?

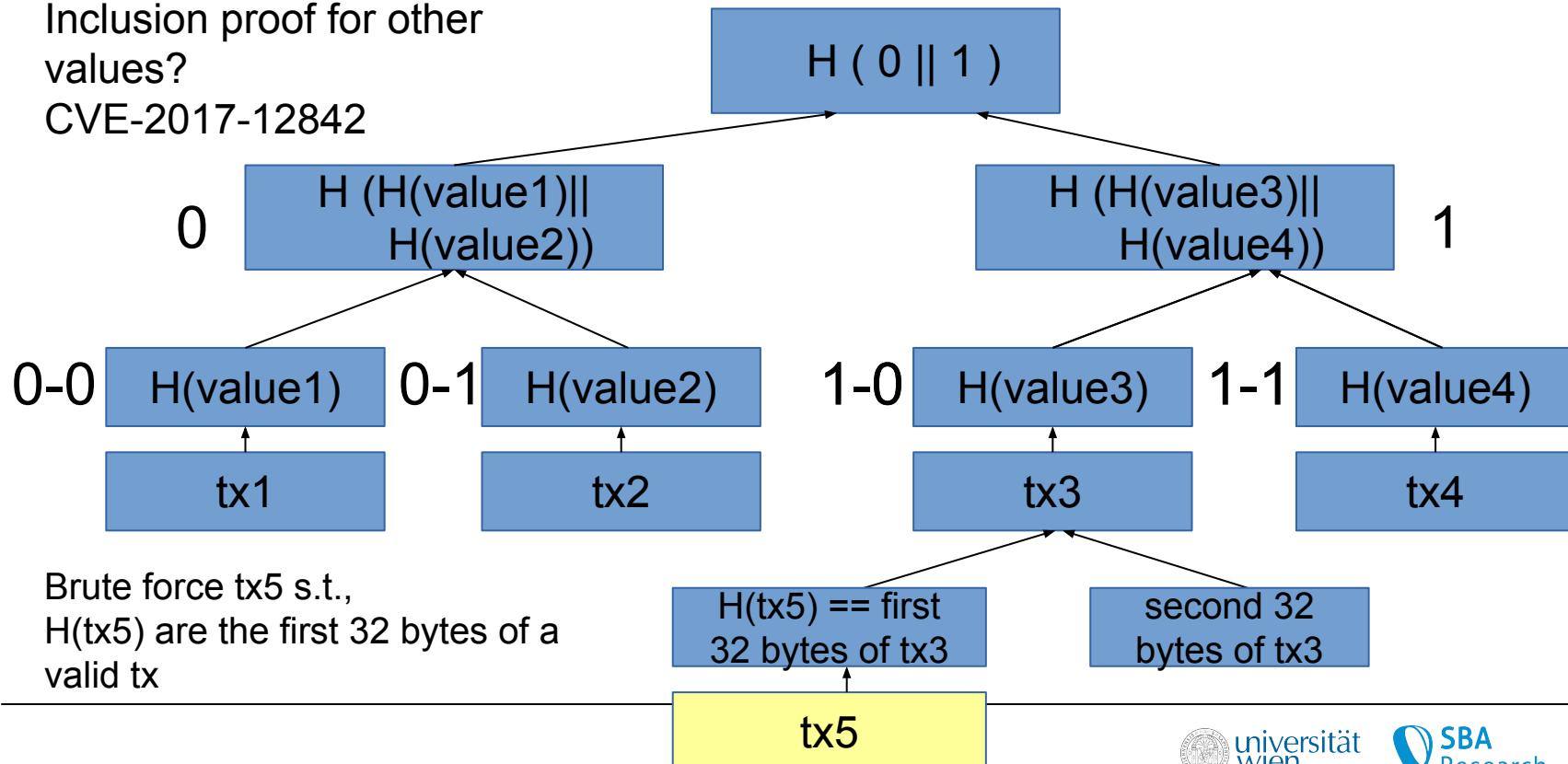
CVE-2017-12842



Merkle trees in Bitcoin

- Inclusion proof for other values?

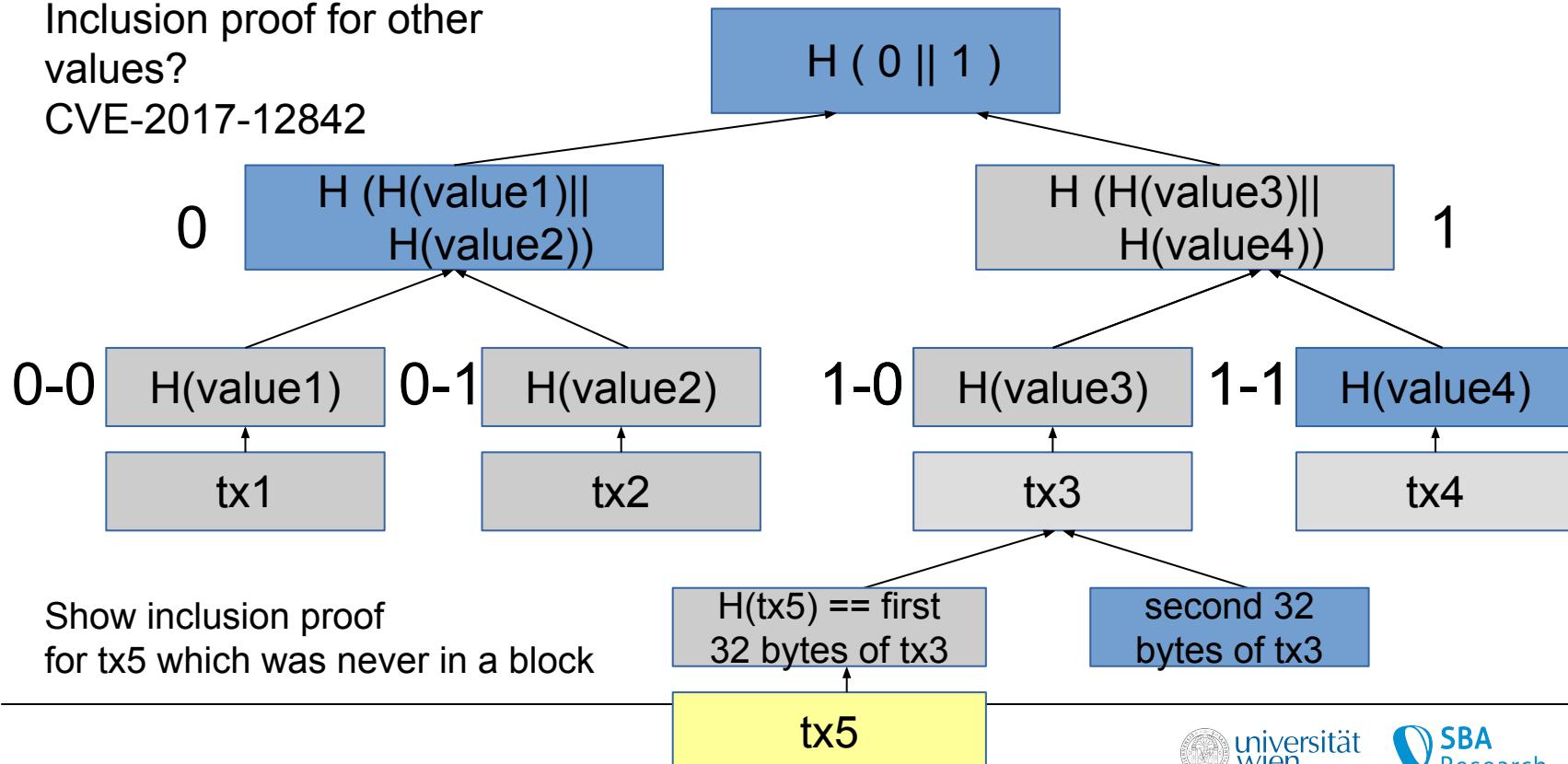
- CVE-2017-12842



Merkle trees in Bitcoin

- Inclusion proof for other values?

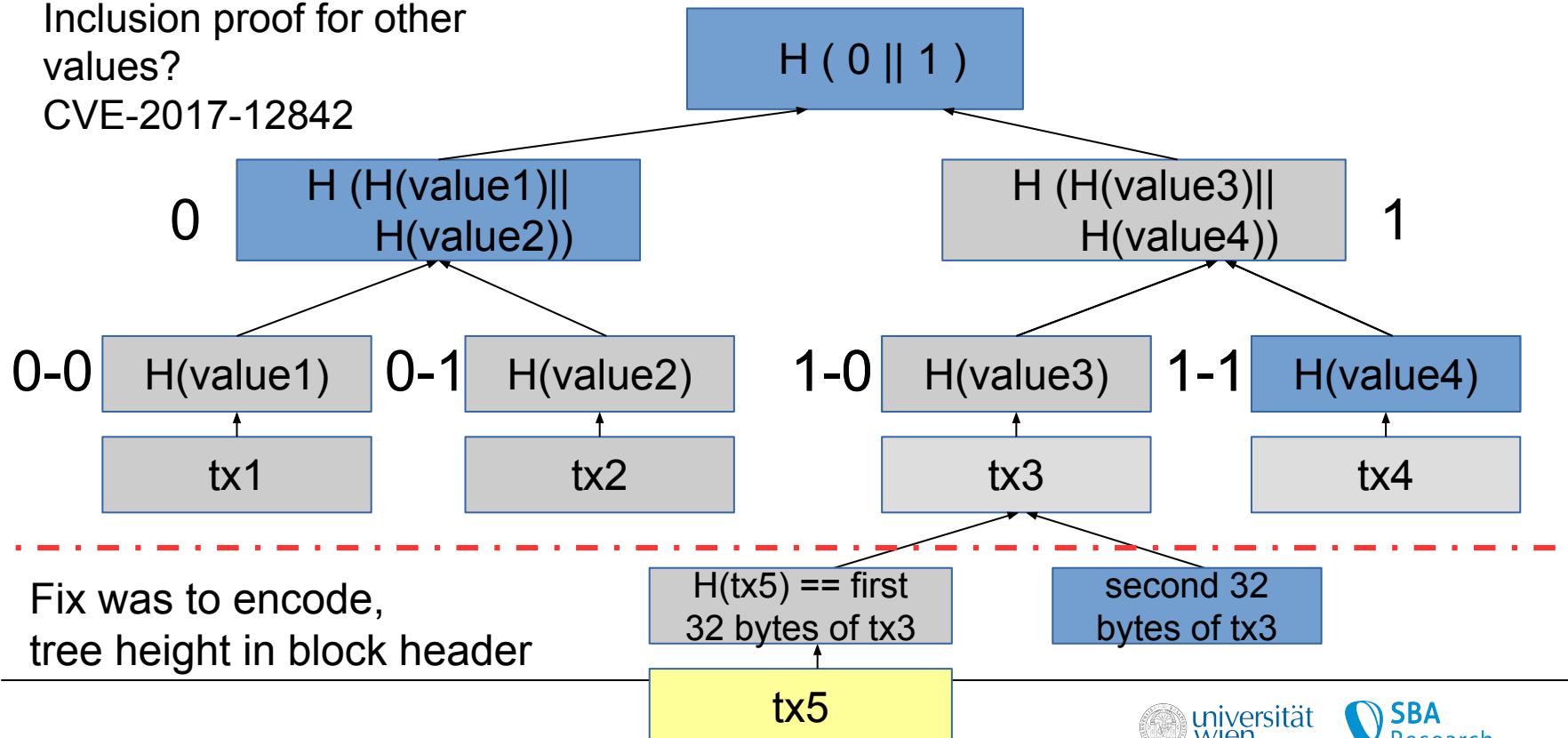
- CVE-2017-12842



Merkle trees in Bitcoin

- Inclusion proof for other values?

CVE-2017-12842



Further information:

- Merkle tree algorithms:
 - Fractal Merkle Tree Representation and Traversal
 - <http://citeserx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.6133&rep=rep1&type=pdf>
- CVE-2017-12842
 - <https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-June/016091.html>
 - <https://bitslog.com/2018/06/09/leaf-node-weakness-in-bitcoin-merkle-tree-design/>
 - <https://github.com/bitcoin/bitcoin/commit/be8651dde7b59e50e8c443da71c706667803d06d>
 - <https://en.bitcoin.it/wiki/CVEs>



Bloom Filter



Bloom filters

- Probabilistic data structure, conceived by Bloom in 1970
- Test whether an element is a member of a set
- **False positives** are possible! False negatives are not
 - Either it might be in the set, or it is *definitely not* in the set

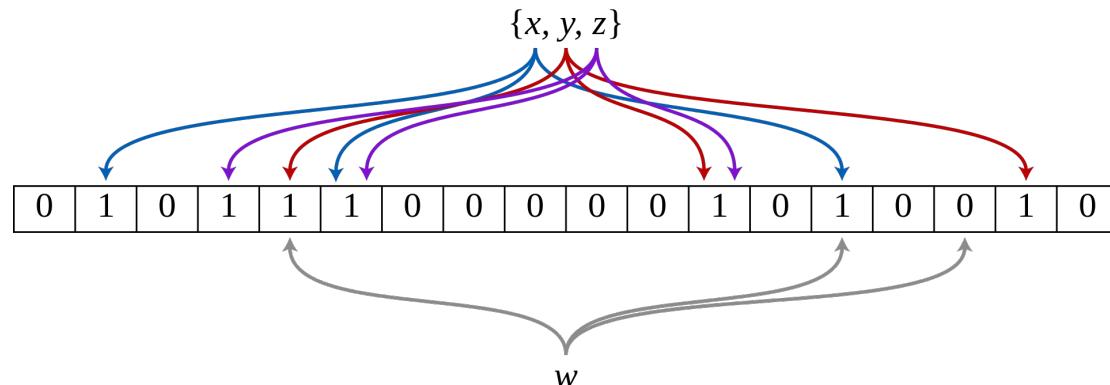
Configure a bloom filter:

- An empty bloom filter is a list of m bits
- Define k different hash functions (in our example $k=3$)
- The number of elements that will be stored in the list denoted by n



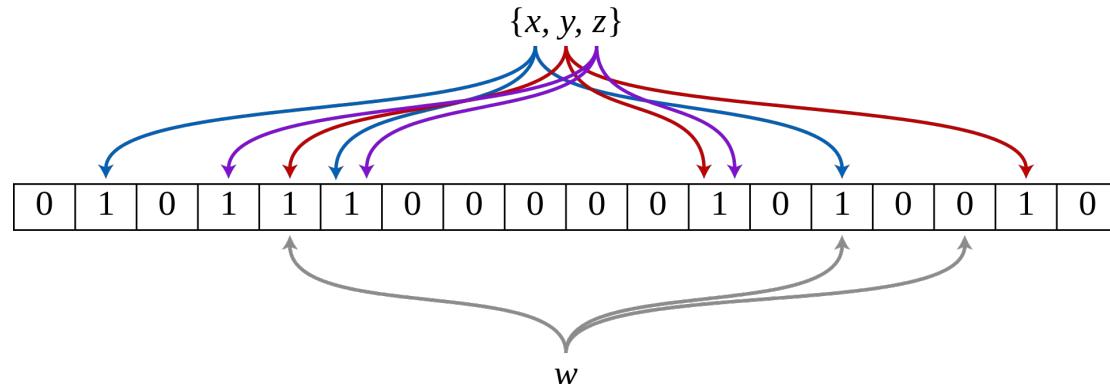
Bloom filters

- Add value:
 - Hash an input value w with each of the k hash functions, for each result map the result to a *single* bit in the list (e.g., mod m), set this bit to 1



Bloom filters

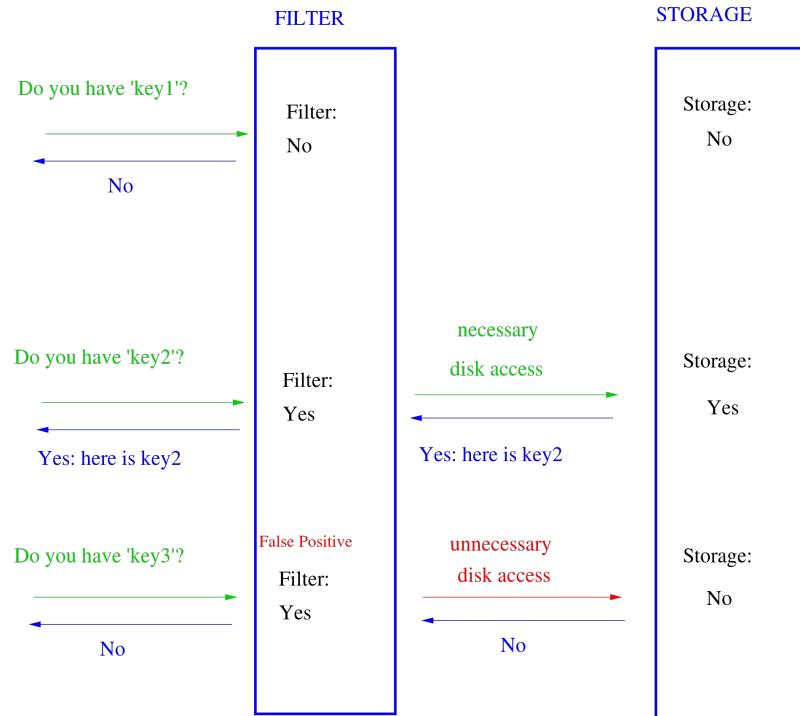
- Check if value is in set:
 - Hash an input value w with each of the k hash functions and map the result to a *single* bit in the list, check if
 - If **all** bits are set to 1 \Rightarrow the value w might be in the set
 - If not all bits are set to 1 \Rightarrow the value w is definitely not in the set



Bloom filters

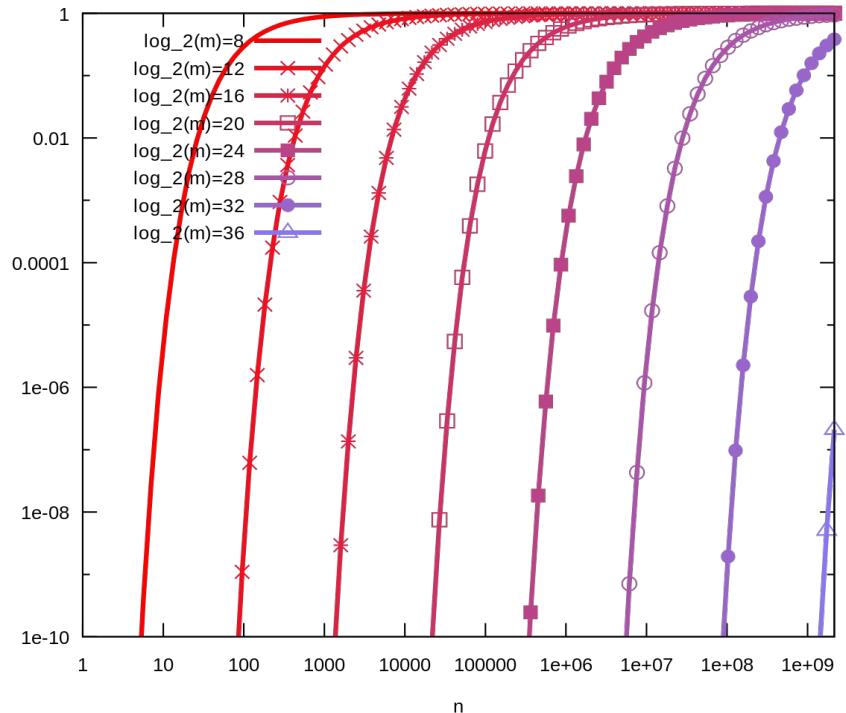
Example usage

- to increase performance
- to commit to but hide the underlying set



Bloom filters

- The error rate/false positive rate can be adjusted by increasing m
 - The optimal number of hash functions is used ($k=(m/n) \ln 2$)
- Example: For storing 100 Million entries with error probability < 1 in a Billion a bloom filter of size 512 MB is needed



Hash Functions & Passwords

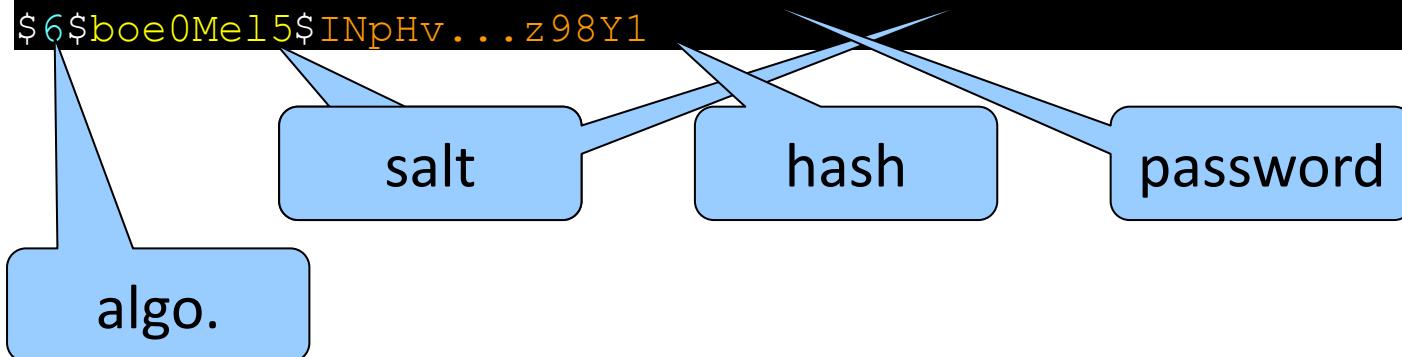


Store passwords

Passwords should only be stored as salted hashes

```
# tail -n1 /etc/shadow  
fox:$6$boe0Me15$INpHv...z98Y1:17975:0:99999:7:::
```

```
# mkpasswd -m SHA-512 notsafeforwork boe0Me15  
$6$boe0Me15$INpHv...z98Y1
```



Store passwords

Passwords should only be stored as salted hashes

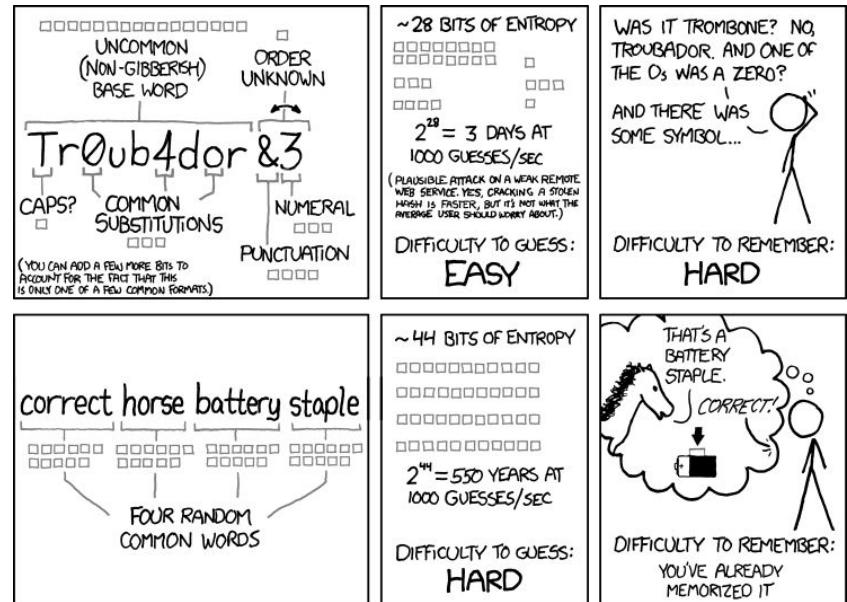
```
# tail -n1 /etc/shadow  
fox:$6$boe0Me15$INpHv...z98Y1:17975:0:99999:7:::  
  
# mkpasswd -m SHA-512 notsafeforwork boe0Me15  
$6$boe0Me15$INpHv...z98Y1
```

Usage of salted passwords increases brute-force efforts if hash leaked.
Rainbow tables (pre calculated password/hash look-up tables) are not possible.

Ideally also the number of rounds i.e., iterated application of hash function, is high (default is 5000)

Good passwords

- the longer the better!
 - generate a different random password for each login
 - use a password manager!
-
- Xkcd:
 - English B1 passive vocabulary
2500 words $\approx 2^{11}$
 - ```
In [26]: math.log2(2_500)
Out[26]: 11.287712379549449
```
    - ```
In [27]: math.log2((2**11)**4)
Out[27]: 44.0
```



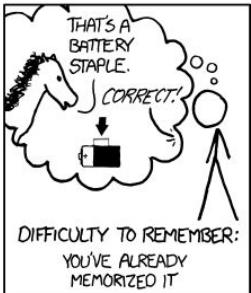
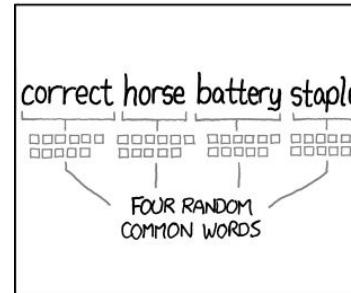
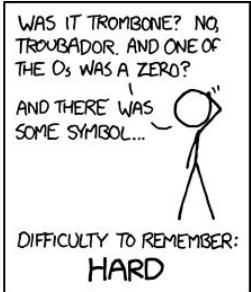
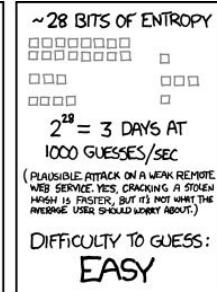
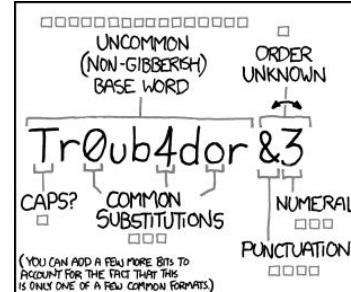
Good passwords

- the longer the better!
- generate a different random password for each login
- use a password manager!
- Xkcd:

- English B1 passive vocabulary
2500 words $\approx 2^{11}$

```
>>> import math  
>>> math.log2(2_500)  
11.287712379549449  
>>> math.log2((2**11)**4)  
44.0
```

- Must still be long enough! E.g., not “a a a a”



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.



Good passwords

- When does the password length exceed the output space of the underlying hash function used to store the digest?
- Assuming base58 char. set 44 characters (input space) are already larger then the output space of a 256 bit hash function
⇒ no point in having a longer password than 44 char.

```
In [29]: import string

In [30]: string.printable

Out[30]: '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!#$%&\'()*+, -./:;=<>?@[\\\]^_`{|}~ \t\n\r\x0b\x0c'

In [31]: chars = string.printable[:-38]

In [32]: chars

Out[32]: '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [33]: len(chars)

Out[33]: 62

In [34]: base58 = chars.replace('0','').replace('0','').replace('l','').replace('1','')

In [35]: base58

Out[35]: '23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

In [36]: len(base58)

Out[36]: 58

In [37]: math.log2(58**44)

Out[37]: 257.7511637856132
```



Password cracking example

- . Bitcoin/Cryptocurrency “Brain Wallet”
 - Idea: Derive ECDSA secret and public key from (memorizable) pass phrase e.g., (roughly):
 - . password := *“how much wood could a woodchuck chuck if a woodchuck could chuck wood”*
 - . secret key d := SHA256(str)
 - . public key pk := dG
 - either compressed (x) or uncompressed (x,y) form
 - . pkhash := RIPEMD160(SHA256(pk))
 - . address := Base58(pkhash)
 - e.g.,: 18s9tQHgcyykBqJkuHaxUad9vrWSXf9aqC

Password cracking example

- Not a good idea!
 - Blockchain = big (unsalted) password hash database
- The Bitcoin Brain Drain [1]
- Brainflayer [2]
 - 130 K guesses/second against entire blockchain (on laptop)
 - GPU/FPGA/ASIC acceleration possible

[1] <https://allquantor.at/blockchainbib/#vasek2016bitcoin>

[2] <https://github.com/ryancdotorg/brainflayer>

Password cracking example

- . Idea of the attack to be efficient:
 - ~ 80,000,000 BTC addresses (n) to one 512 MB Bloom filter (m)
 - match each address to ~ 20 locations in bitmask (k)
 - check against all addresses at once with output
 - . probable match => slower check to identify false positives
 - . certainly no match



Password cracking example

Successfully cracked:

“how much wood could a woodchuck chuck if a woodchuck could chuck wood”

- ~ 250 BTC

“”

- ~ 50 BTC

“Down the Rabbit-Hole”

- ~ 85 BTC

“The Quick Brown Fox Jumped Over The Lazy Dot”

- ~ 85 BTC

Password cracking example

“gate gate paragate parasamgate bodhi svaha”

“The Persistence Of Memory”

“QTC”

“644122178”

“8964009”

“que me lleve la muerte”

“one two three four five six seven”

“it’s a secret to everybody”

“Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn”

“my hovercraft is full of eels”

“Interior Crocodile Alligator”

“No need to worry, my accountant handles that”

“tomb-of-the-unknown-soldier-identification-badge”

“permit me to issue and control the money of a nation and i care not who makes its laws”

“who is john galt”

“Live as if you were to die tomorrow. Learn as if you were to live forever.”



Message Authentication Codes (MACs)



Message Authentication Codes

- Informally, a message authentication code uses a *secret* and a *message* to produce a *tag* for a message
- The tag can be created and verified only if the secret is known!
- Goals:
 - **Integrity:** The modification of the message should be prevented if message and tag are transmitted in the clear
 - **Authenticity:** If the secret is shared only between two parties the message is also authenticated



CBC-MAC

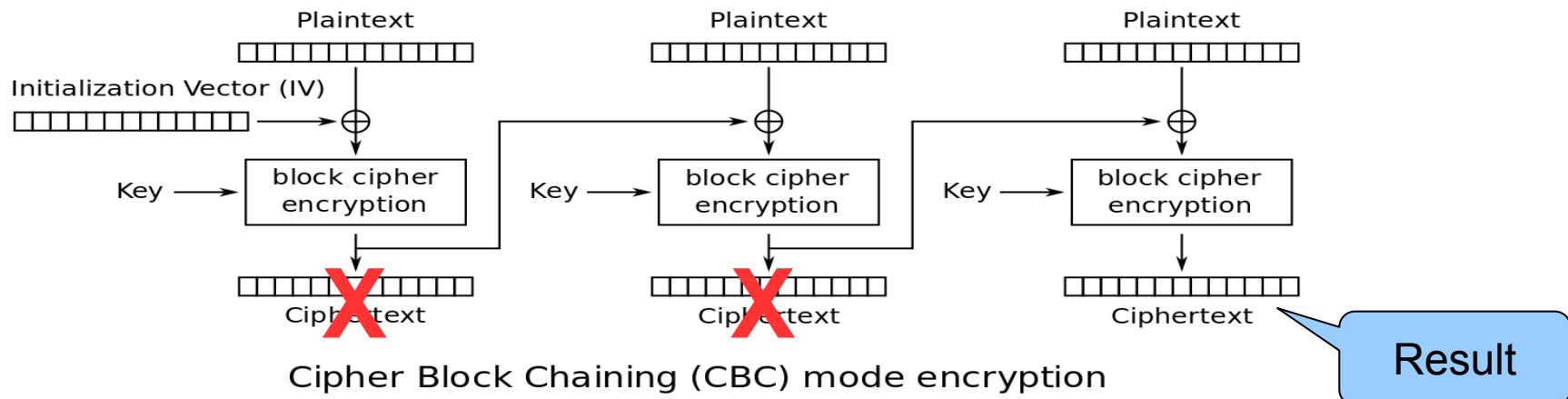
- Cipher Block Chaining Message authentication code (CBC-MAC)
- Should ensure integrity of transmitted data
- Calculation based on a *shared secret* so that it cannot be forged by someone who does not know the key
- When used stand-alone, there is *no-encryption* and hence no confidentiality
- Advantage: It uses CBC mode for construction and verification and if there already is a block cipher implementation in a resource constrained environment (e.g. embedded system) this can be reused



CBC-MAC

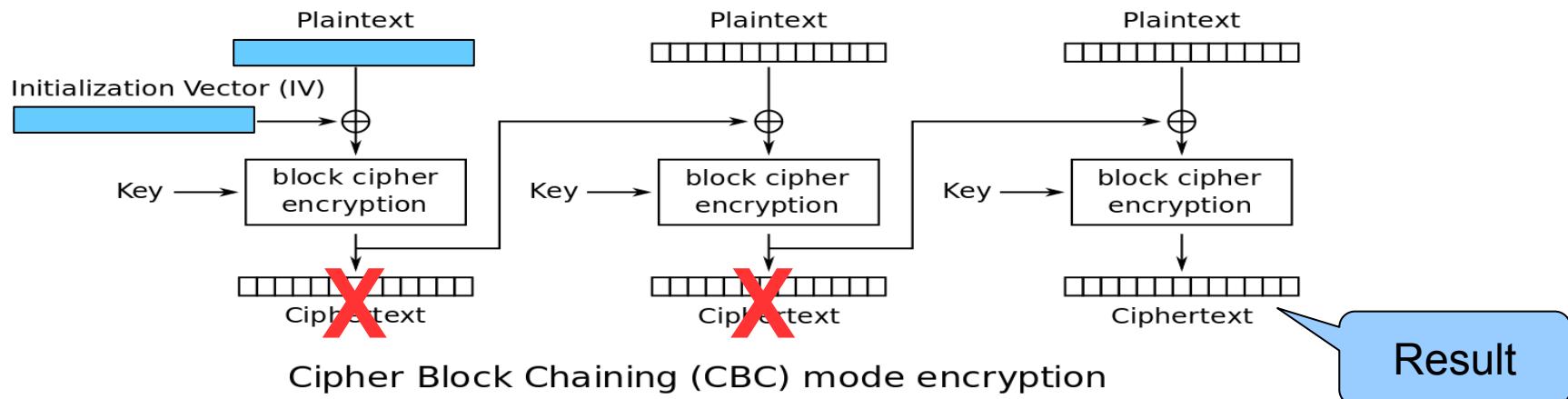
Cipher Block Chaining Message authentication code (CBC-MAC)

- Use CBC and discard all blocks despite the last
- Creation and verification is the same operation, i.e. encryption



CBC-MAC

- What if CBC is used to verify integrity plain text transmitted unencrypted over the wire together with the IV?
- Active attack; KPA



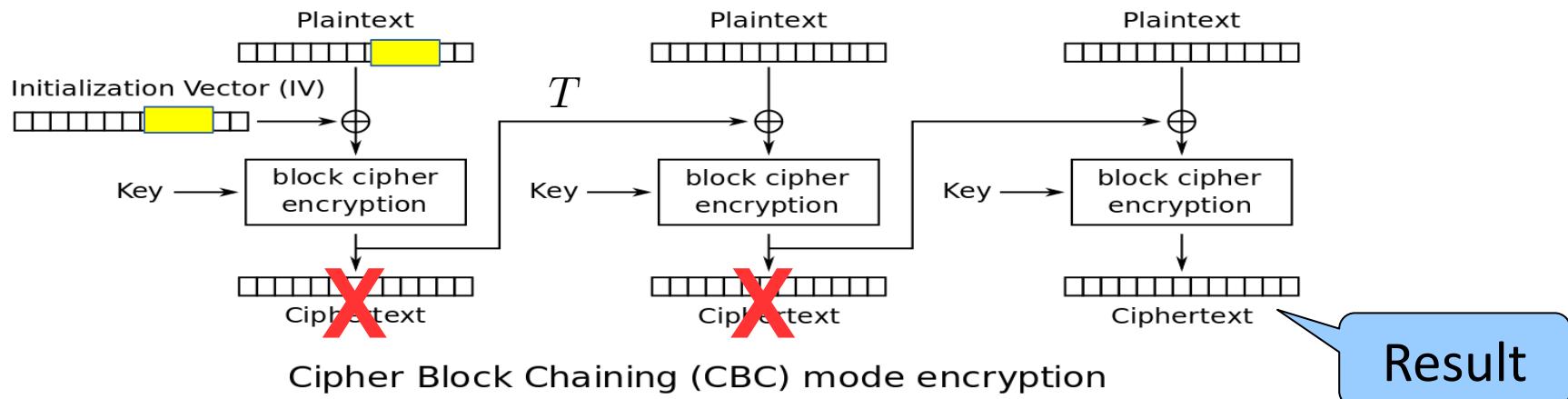
CBC-MAC

- By changing the IV and the first block
- any change to the first block can be canceled out

$$(IV \oplus p_0) = T$$

$$(IV' \oplus p'_0) = T'$$

$$T = T'$$



Message Authentication Codes

- Is a hash function used like that a good MAC?

$$tag = H(secret \parallel message)$$

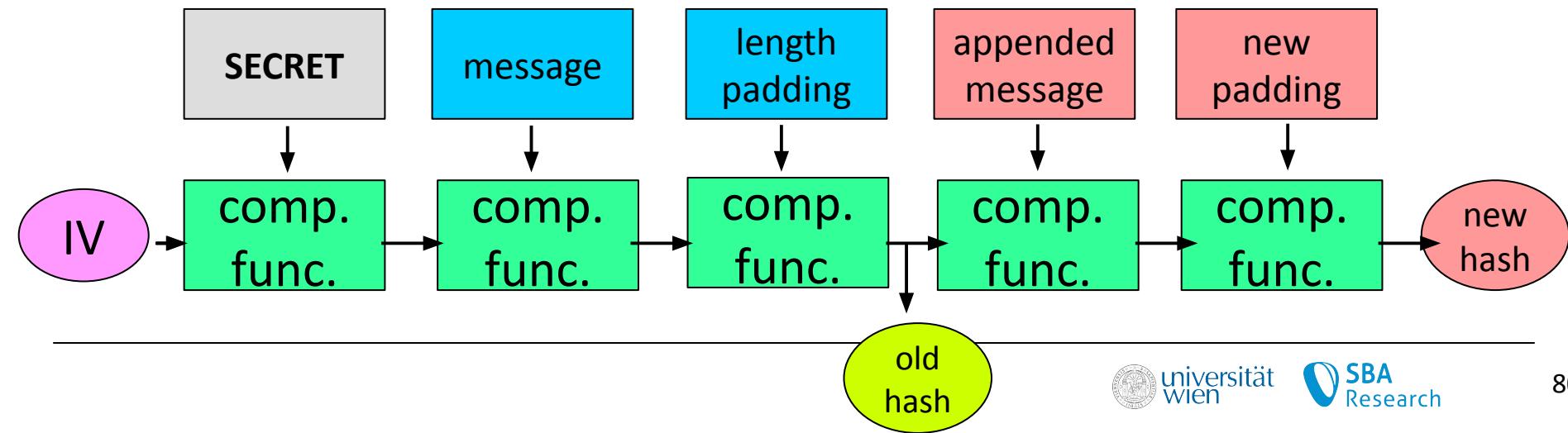
- Then the tag and message are sent over the wire unencrypted
- Is the integrity of the message ensured against an active attacker?

Hash length extension attack

- . In a hash length extension attack an attacker can use $H(\text{message_1})$ and the length of message_1 to calculate $H(\text{message_1} \parallel \text{message_2})$ for a message_2 chosen by the attacker.
- . All algorithms based on the *Merkle-Damgård* construction are susceptible to this kind of attack
 - e.g., MD5, SHA-1, SHA-2
- . That does not mean that these algorithms are broken! Just that you should not use them directly as MAC (Message Authentication Code) to ensure the integrity and authenticity of some message

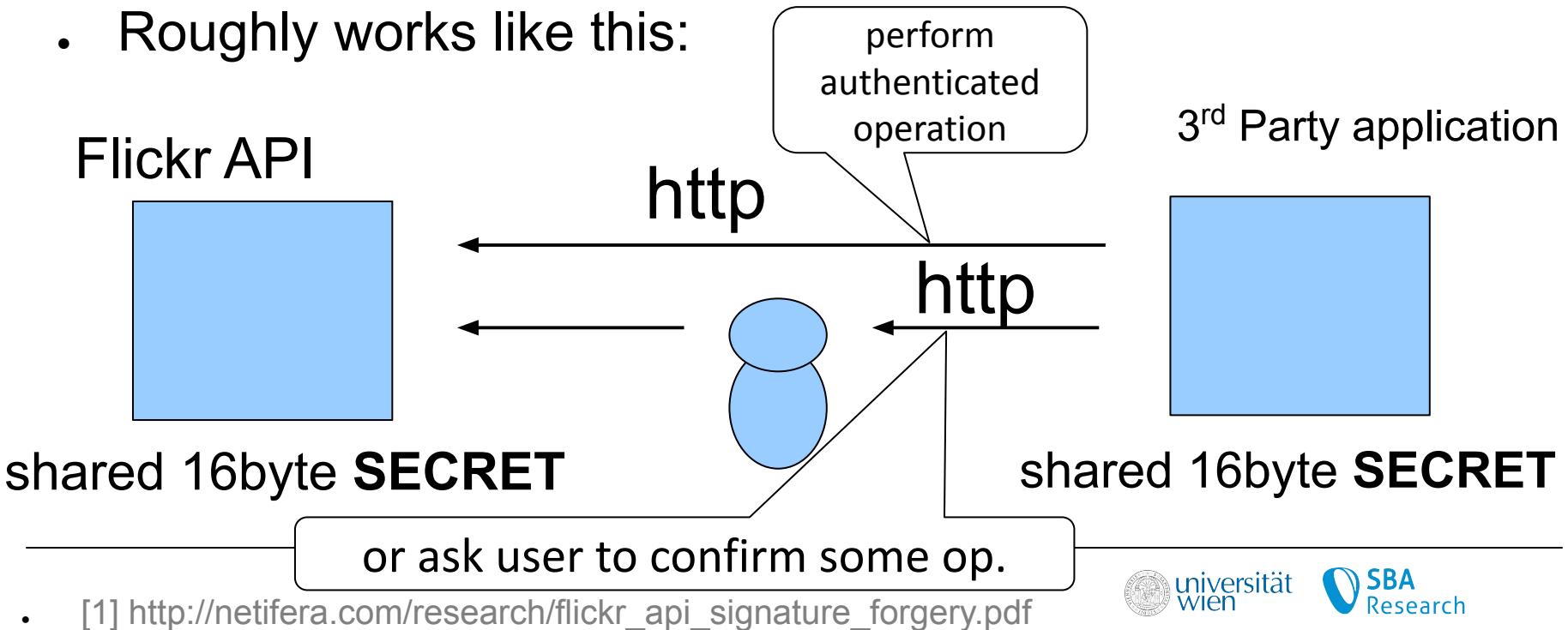
Hash length extension attack

- *Merkle-Damgård construction*
 - chain of *compression functions* and Initialization Vector
 - used in MD5, SHA1, SHA256, SHA512



Hash length extension attack

- Flickr's API Signature Forgery Vulnerability [1]
- Roughly works like this:



• [1] http://netifera.com/research/flickr_api_signature_forgery.pdf

Hash length extension attack

Request:

```
http://www.flickr.com/services/auth/?  
api_key=44fefafa051fc1c61f  
&perms=read  
&api_sig=1a947ced7375aadf970ccd7ee2bb792b81d5ed1c
```

Resulting hash function input:

```
SECRETapi_key44fefafa051fc1c61fpermsread
```

```
$ echo -n "SECRETapi_key44fefafa051fc1c61fpermsread" | shasum  
1a947ced7375aadf970ccd7ee2bb792b81d5ed1c
```

Hash length extension attack

- Construct new request using *hashpump*[1]
 - Takes signature (i.e., old hash)
 - detects hash format
 - supported MD5, SHA1, SHA256, SHA512
 - Takes old input data after SECRET
 - Takes key length (i.e., length of SECRET)
 - Takes message/data to add
 - Calculates new padding automatically
 - Calculates new valid signature / hash
-
- [1]<https://github.com/bwall/HashPump>

```
$ ./hashpump
Input Signature: 1a947ced7375aadf970ccd7ee2bb792b81d5ed1c
Input Data: api_key44fefafa051fc1c61fpermsread
Input Key Length: 6
Input Data to Add: api_key44fefafa051fc1c61fpermsswriteactionevil
f43a9343fd2ee59dc43cf32ce52885a85753f8
api_key44fefafa051fc1c61fpermsread\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x010api_key44fefafa051fc1c61fpermsswriteactionevil
```

Check if signature is valid with correct SECRET:

```
$ perl -e 'print
"SECRETapi_key44fefafa051fc1c61fpermsread\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x010 api_key44fefafa051fc1c61fpermsswr
iteactionevil"' | shasum
f43a9343fd2ee59dc43cf32ce52885a85753f8
```



Hash length extension attack

- . Details:
 - For the exploit to be practical they needed to supply the old http variables without actually changing the new variables by adding the new variable ‘a=’ to the beginning
 - Then concatenated input to hash function would stay the same but application logic does not use unknown variables

```
http://www.flickr.com/services/auth/?  
a=pi_key44fefa051fc1c61fpermsread [padding]  
&api_key=44fefa051fc1c61f  
&perms=write  
&action=evil
```

Message Authentication Codes

- Solutions:
 - use other hash function not vulnerable to hash length extension attack e.g., SHA3
 - use HMAC as a MAC which is not vulnerable to length extension attacks by construction
 - . roughly constructed like that
 - . ipad and opad are fixed constants (must not be equal)

$$HMAC(K, m) = H\left((K' \oplus opad) || H((K' \oplus ipad) || m)\right)$$



How to use MAC?

- **MAC-then-Encrypt**
 - Compute MAC on data, append tag to data, then encrypt the whole
- **MAC-and-Encrypt**
 - Compute MAC on data, encrypt the data, then append the tag at the end of the ciphertext
- **Encrypt-then-MAC**
 - Encrypt the data, then compute the MAC on the ciphertext, then append tag to the ciphertext



How to use MAC?

- **MAC-then-Encrypt**
 - Compute MAC on data, append tag to data, then encrypt the whole
 - Does not provide integrity of ciphertext. Must be decrypted before check, might be a problem if ciphertext is malleable (i.e. there might be more than one correct version)
- **MAC-and-Encrypt**
 - Compute MAC on data, encrypt the data, then append the tag at the end of the ciphertext
 - Again problem if ciphertext is malleable
 - May reveal information from the plaintext => same plaintexts yield same MACs
- **Encrypt-then-MAC**
 - Encrypt the data, then compute the MAC on the ciphertext, then append tag to the ciphertext
 - Provides integrity of ciphertext & plaintext without leaking plaintext information



From Symmetric Cryptography To Asymmetric Cryptography



Can you create public-key cryptosystem and a signature scheme by only using symmetric primitives?



Can you create public-key cryptosystem and a signature scheme by only using symmetric primitives?

Yes, in an inefficient ways



Merkle Puzzles

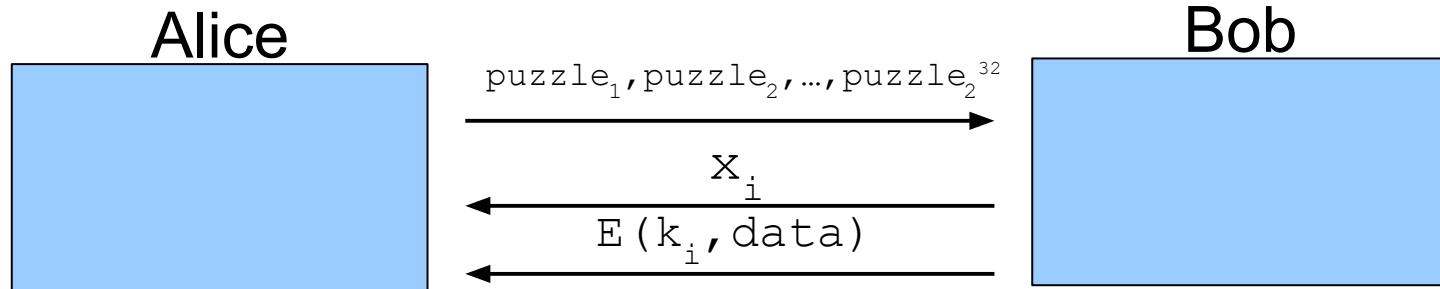
One of the first constructions for a asymmetric cryptosystem

- Devised by Ralph Merkle in 1974 (published 1978)
- A basic key exchange protocol
- Threat model: passive attacker

[1] <http://www.itas.kit.edu/pub/m/2002/mewe02a.htm>

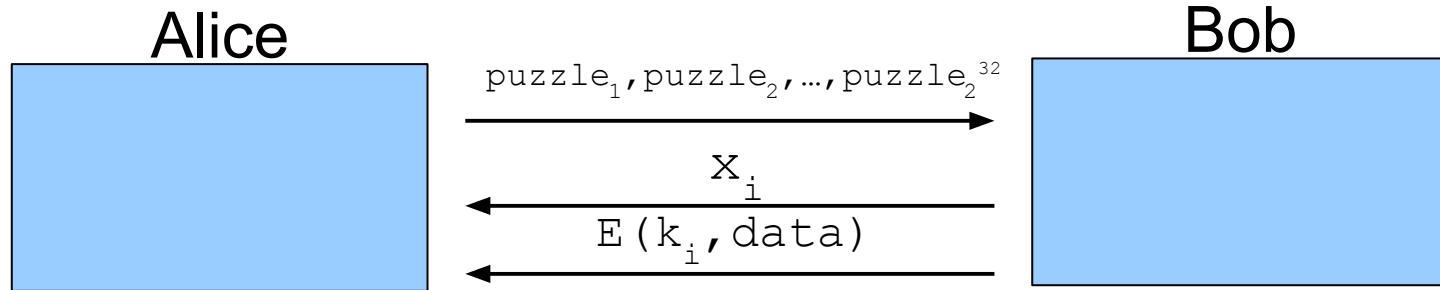
[2] <https://crypto.stanford.edu/~dabo/courses/OnlineCrypto/>

Merkle Puzzles



- $E(k, m)$ is a symmetric cipher where $k \in \{0, 1\}^{128}$
- p_i is a key in which the first 96 bits are set to 0 s.t., there are 2^{32} possibilities
- Alice prepares 2^{32} puzzles with random indices (x_i) and keys (k_i) s.t., $\text{puzzle} = E(p_i, \text{"Puzzle } x_i\text{"} || k_i)$
- Bob chooses a random puzzle and brute forces p_i
- Once successful he sends the decrypted index x_i to Alice and encryptes with k_i

Merkle Puzzles



- Effort Alice, prepare n puzzles: $O(n)$
In our case $n = 2^{32}$
- Effort Bob, solve one puzzle: $O(n)$
- Effort Attacker, worst case solve all puzzles: $O(n^2)$
(i.e., 2^{64} worst case, 2^{63} on average)
- Quadratic gap is best possible if cipher modelled as black box [1]

[1] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations.
In Proceedings of the Symposium on Theory of Computing (STOC), pages 44–61, 1989.

Lamport Signatures

A method for creating a digital signature

- Signature can be verified against a previously exchanged verification key (vk)
- Under the assumptions that vk can be transmitted to the verifier *without modification*, a one-way function and a way to generate random numbers is enough to create a signature scheme.

Lamport Signatures

- Requires one-way function $f(x)$
(for practical purposes think of SHA256)
- Requires a way to securely generate random numbers
- If you want to sign a 256 bit value, signer needs to create $256 \cdot 2$ keys
- Two secret/public- keys for every bit of the message
- The signature are the preimages to the individual bits m_1, m_2, \dots, m_{256} ,
of the message m

$$sk = [\langle s_{1_0}, s_{1_1} \rangle, \langle s_{2_0}, s_{2_1} \rangle, \dots, \langle s_{256_0}, s_{256_1} \rangle] \quad (1)$$

$$vk = [\langle f(s_{1_0}), f(s_{1_1}) \rangle, \langle f(s_{2_0}), f(s_{2_1}) \rangle, \dots, \langle f(s_{256_0}), f(s_{256_1}) \rangle] \quad (2)$$

$$\sigma = [s_{1_{m_1}}, \dots, s_{2_{m_{256}}}] \quad (3)$$



Lamport Signatures

- Requires one-way function $f(x)$
(for practical purposes think of SHA256)
- Requires a way to securely generate random numbers
- If you want to sign a 256 bit value, signer needs to create $256 \cdot 2$ keys
- Two secret/public- keys for every bit of the message
- The signature are the preimages to the individual bits m_1, m_2, \dots, m_{256} ,
of the message m

$$sk = [\langle s_{1_0}, s_{1_1} \rangle, \langle s_{2_0}, s_{2_1} \rangle, \dots, \langle s_{256_0}, s_{256_1} \rangle] \quad (1)$$

$$vk = [\langle f(s_{1_0}), f(s_{1_1}) \rangle, \langle f(s_{2_0}), f(s_{2_1}) \rangle, \dots, \langle f(s_{256_0}), f(s_{256_1}) \rangle] \quad (2)$$

$$\sigma = [s_{1_{m_1}}, \dots, s_{2_{m_{256}}}] \quad (3)$$

How many messages can I
sign with one pk?

Lamport Signatures

- Requires one-way function $f(x)$
(for practical purposes think of SHA256)
- Requires a way to securely generate random numbers
- If you want to sign a 256 bit value, signer needs to create $256 \cdot 2$ keys
- Two secret/public- keys for every bit of the message
- The signature are the preimages to the individual bits m_1, m_2, \dots, m_{256} ,
of the message m

$$sk = [\langle s_{1_0}, s_{1_1} \rangle, \langle s_{2_0}, s_{2_1} \rangle, \dots, \langle s_{256_0}, s_{256_1} \rangle] \quad (1)$$

$$vk = [\langle f(s_{1_0}), f(s_{1_1}) \rangle, \langle f(s_{2_0}), f(s_{2_1}) \rangle, \dots, \langle f(s_{256_0}), f(s_{256_1}) \rangle] \quad (2)$$

$$\sigma = [s_{1_{m_1}}, \dots, s_{2_{m_{256}}}] \quad (3)$$

How many messages can I
sign with one pk? => **one**

Lamport Signatures

If the message to be signed is random, each additional signature halves the security level i.e., attacker learns more bits

If the message is chosen by the attacker, two signatures of messages where each bit differs are enough to create an arbitrary signature

Signatures and keys are “large”:

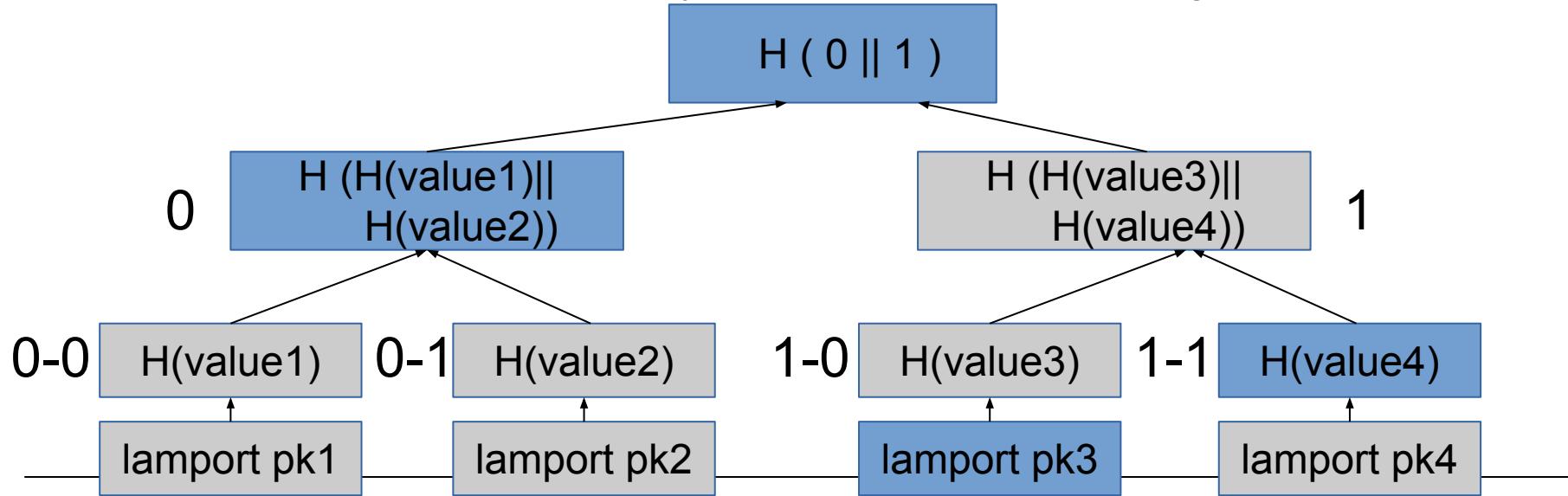
- vk size is $2 \cdot 256 \cdot 256 = 128$ Kibit
- signature size is $256 \cdot 256 = 64$ Kibit

Several optimizations and improvements to Lamport signatures have been proposed.



Lamport Signatures

- Merkle trees have been invented by Ralph Merkle to transform Lamport one-time signatures to a more useable scheme. Signature contains Lamport public keys plus inclusion proof e.g.,



[1] <https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf>

Hash based signatures

Currently Merkle trees in connection with hash based signatures become more relevant again:

- Assumed to survive quantum computer
- Post Quantum Computer (PQC) cryptography challenge by NIST:
candidate: SPHINCS+

Basically, Lamport sigs. and Merkle trees on steroids i.e., layered trees so large that you can randomly chose a branch to a “one-time” key and be safe without keeping records which have already been used

<http://sphincs.org/>

<https://sphincs.cr.yp.to/index.html>

Post-Quantum Cryptography

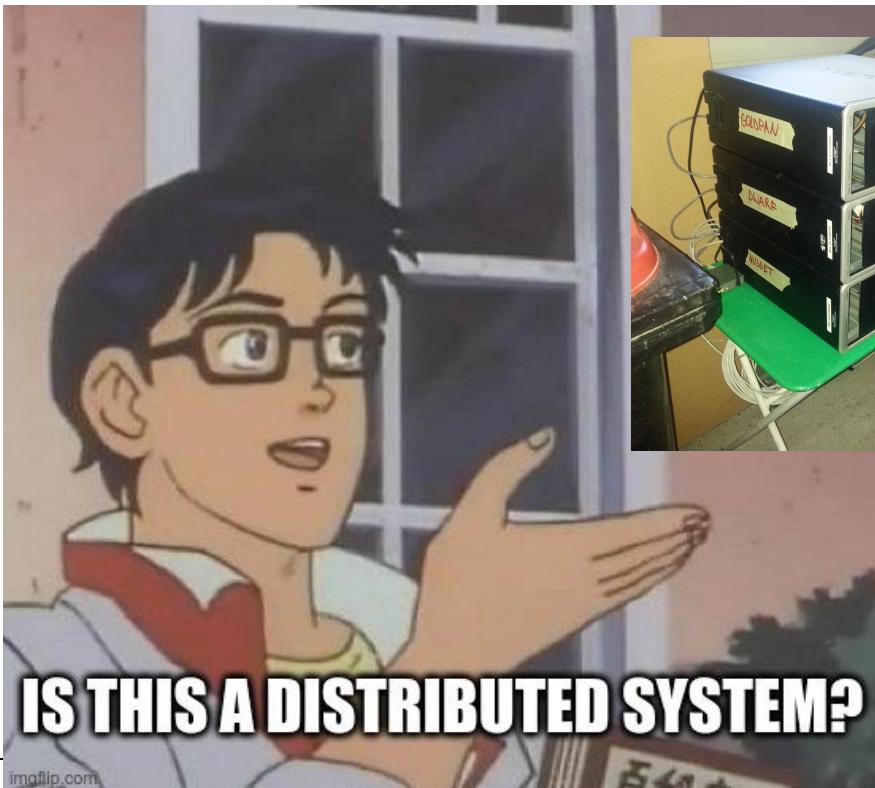
NIST Post-Quantum Cryptography Standardization

- started 2017 with 23 signature and 59 encryption/KEM schemes
- currently in its final phase
- Criticism issued by Dan Bernstein that selection has not been 100% transparent and potentially biased towards Kyber
- More info:
 - <https://blog.cr.yp.to/20231003-countcorrectly.html>
 - <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography>
 - <https://pqcrypto.org/index.html>

(Informal) Introduction to Distributed Computing and Distributed Systems



What is a Distributed System?



universität
wien

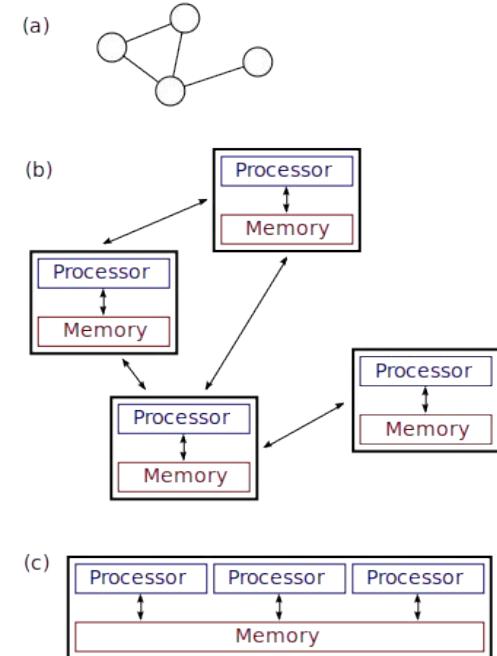
SBA
Research

What is a Distributed System?

- Let's first ask Wikipedia:

„A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another.

Distributed computing is a field of computer science that studies distributed systems.“



a,b: distributed system
c: parallel system

What is a Distributed System?

- Tanenbaum and Steen:

“Various definitions of distributed systems have been given in the literature, none of them satisfactory, and none of them in agreement with any of the others.”

“A distributed system is a collection of independent computers that appears to its users as a single coherent system.”

- K. Birman:

“Reduced to the simplest terms, a *distributed computing system* is a set of computer programs, executing on one or more computers*, and coordinating actions by exchanging *messages*.”

*emphasis added

ChatGPT (3.5) on the Definition and Defining Properties of a Distributed System

„Distributed Systems: A network of multiple interconnected computers that work together to achieve a common goal, sharing resources and information across the network.“

- **Multiple Nodes:** Comprises multiple interconnected nodes or machines.
- **Autonomy:** Each node operates independently and has its own memory and processing capabilities.
- **Communication:** Nodes communicate with each other by message passing.
- **Concurrency:** Tasks may run concurrently on different nodes.
- **Lack of a Global Clock:** No synchronized global time reference.
- **Failure Independence:** Nodes can fail independently without affecting the entire system.
- **Scalability:** Can be easily scaled by adding or removing nodes.
- **Transparency:** Should appear to users and applications as a single coherent system.

What is a Distributed System?

There has been considerable debate over the years about what constitutes a distributed system. It would appear that the following definition has been adopted at SRC*:

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

Leslie Lamport

So What is a Distributed System?

→ It depends on who you ask

Properties of Distributed Systems we are particularly interested in:

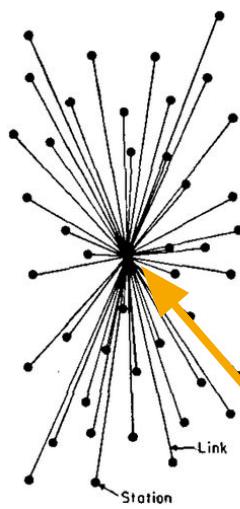
- Independent resources e.g. Memory and CPU
- Communication through message passing
- Some form of coordination to reach common goal(s)



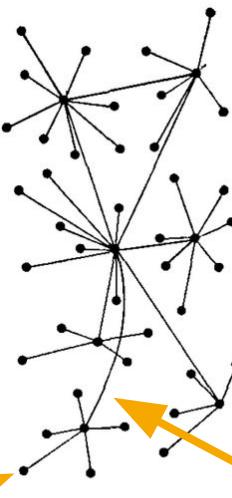
Types of Distributed Systems



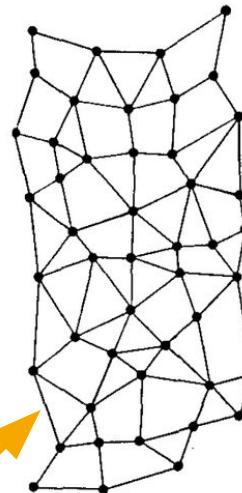
Examples of different types of Distributed Systems



Node*



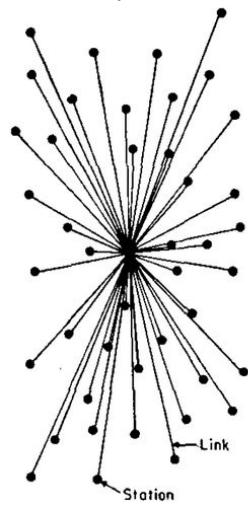
Link / Channel



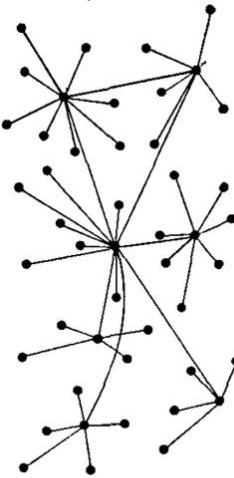
* Also referred to as a **process** depending on the context

(Real-World) Examples of different types of Distributed Systems ?

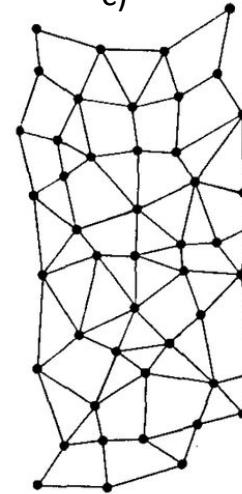
a)



b)



c)



•
•
•

•
•
•

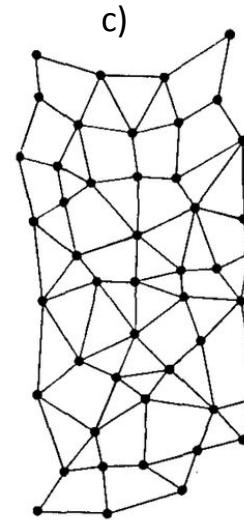
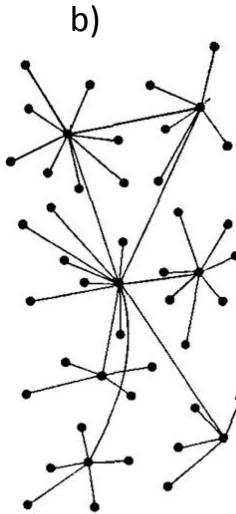
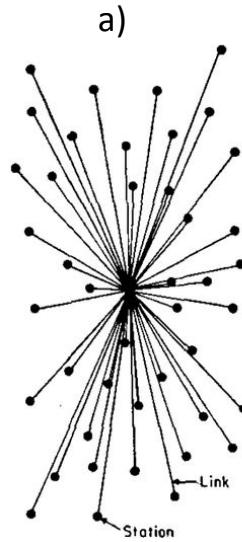
•
•
•



universität
wien

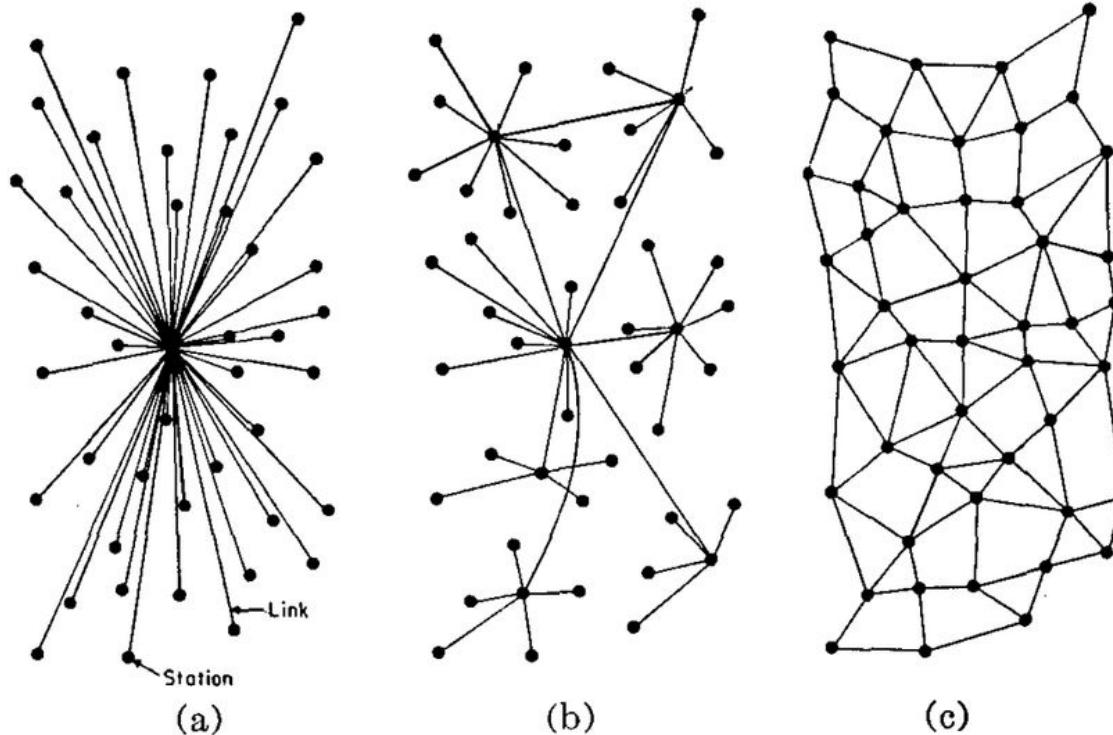
SBA
Research

(Real-World) Examples of different types of Distributed Systems ?

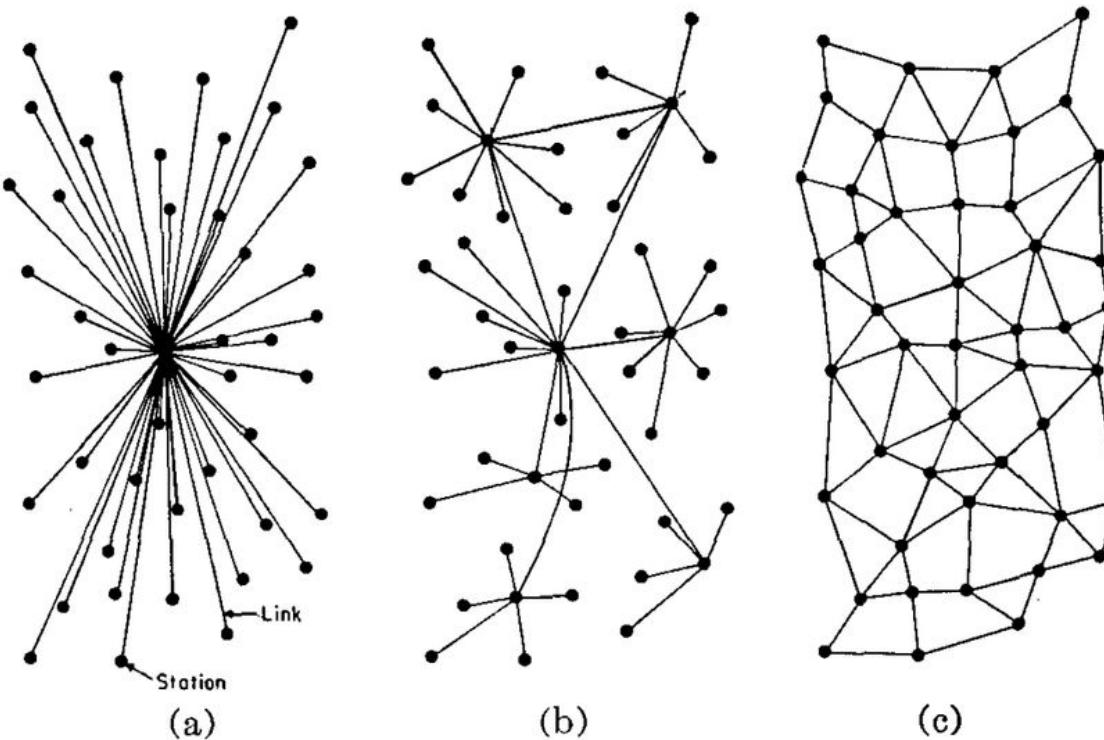


- Webserver
- Database w.passive replicas
- Botnet
- Federated Networks
(EMail, Matrix)
- Modern Internet
- P2P Networks*
- Mixnets? (Tor, I2P)
- Mesh Networks (ZigBee, Arpanet)

*in practice many p2p networks use some form of supernodes, so they actually look more like b)



In terms of their distribution how would you label a), b) and c) ?



(a) Centralized. (b) Decentralized. (c) Distributed networks.

Note: I would disagree with this labeling by Ehmke et al. and swap b) and c)

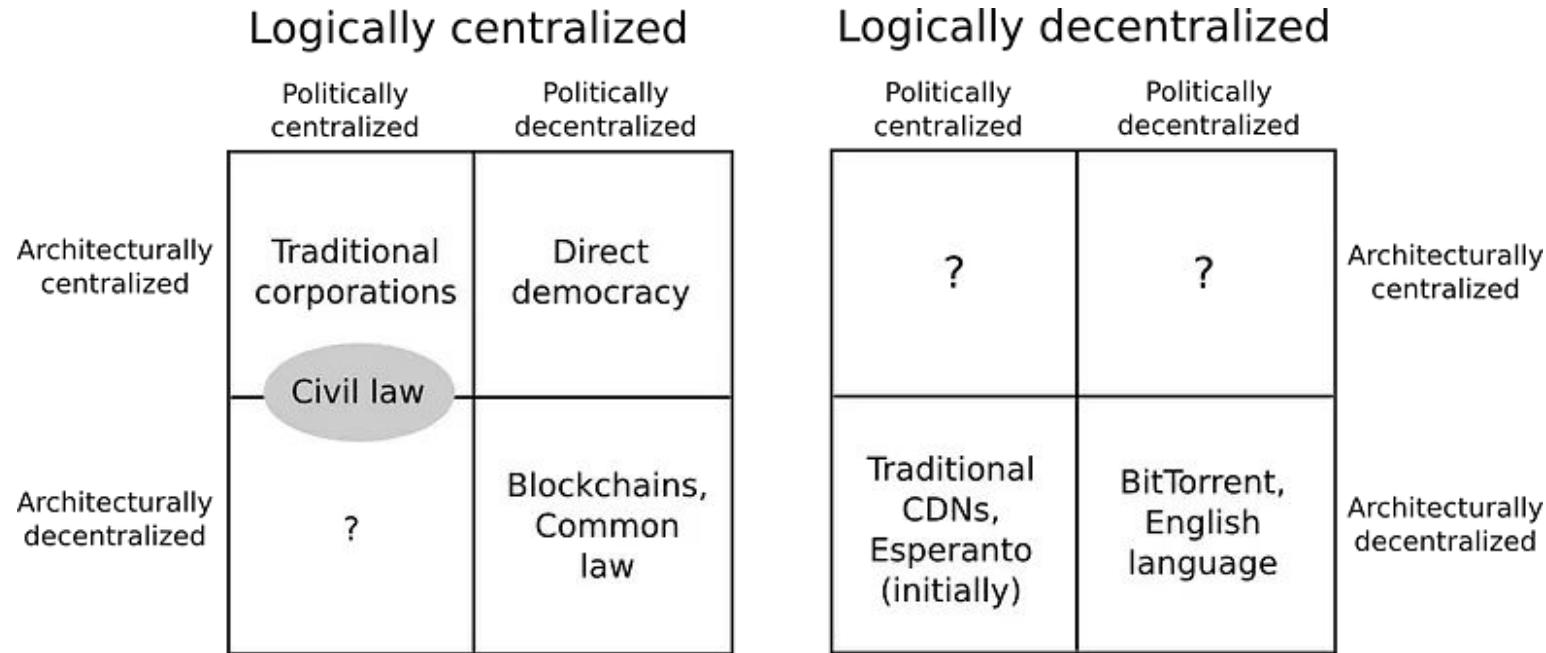
Decentralization in Distributed Systems

"The key difference between distributed systems and decentralized systems is one of authority and trust between components. Differences in architecture and use of security and privacy controls stem from it."

In distributed, but not decentralized, systems the existence of a single authority that provisions and manages all components that are trusted enables the use of simple security, many times based on dedicated trusted components that act as roots of trust.

In decentralized systems no single authority can provision a root of trust or trusted computing base, making security mechanisms reliant on those (such as central access control or traditional public key infrastructures) inapplicable.

Three Axes of Decentralization according to Vitalik Buterin*



*Buterin notes that these placements are very rough and highly debatable

Difficulties in Distributed Computing



"Random communication delays and faults prevent distributed processes from having the instantaneous knowledge of the system state that shared storage provides to the processes of a centralized system. This is one of the main reasons behind the perceived difficulty of programming distributed systems."

[Cristian et al. *Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement*, 1986]

*emphasis added



Difficulty: Random Faults



Difficulty: Random Faults



Random faults can occur in different components, potentially leading to failures

Definitions of Fault and Failure as by IEEE[1]

fault. (1) A defect in a hardware device or component; for example, a short circuit or broken wire.
(2) An incorrect step, process, or data definition in a computer program.

Note : This definition is used primarily by the fault tolerance discipline. In common usage, the terms “error” and “bug” are used to express this meaning.

failure. The inability of a system or component to perform its required functions within specified performance requirements.

Note: The fault tolerance discipline distinguishes between a human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error)

fault tolerance. (1)The ability of a system or component to continue normal operation despite the presence of hardware or software faults.
(2) The number of faults a system or component can withstand before normal operation is impaired.
(3) Pertaining to the study of errors, faults, and failures, and of methods for enabling systems to continue normal operation in the presence of faults.

[1] "IEEE Standard Glossary of Software Engineering Terminology," in *IEEE Std 610.12-1990*, vol., no., pp.1-84, 31 Dec. 1990, doi: 10.1109/IEEESTD.1990.101064.

Definitions of Fault and Failure as by IEEE[1]

fault. (1) A defect in a hardware or component; for example, a short circuit or broken wire. (2) An incorrect step, procedure, or data definition in a computer program.

Note : This definition is used primarily by the fault tolerance discipline. In common usage, the terms "error" and "bug" are used to express this meaning.



failure. (1) The ability of a component to continue operation despite the presence of faults.

(2) The number of faults a system or component can withstand before operation is impaired.

(3) According to the study of errors, failures, and of methods for systems to continue normal in the presence of faults.

Please be aware that the terms **fault** and **failure** are often used loosely interchangeable and depend on the context

[1]"IEEE Standard Glossary of Software Engineering Terminology," in *IEEE Std 610.12-1990*, vol., no., pp.1-84, 31 Dec. 1990, doi: 10.1109/IEEESTD.1990.101064.

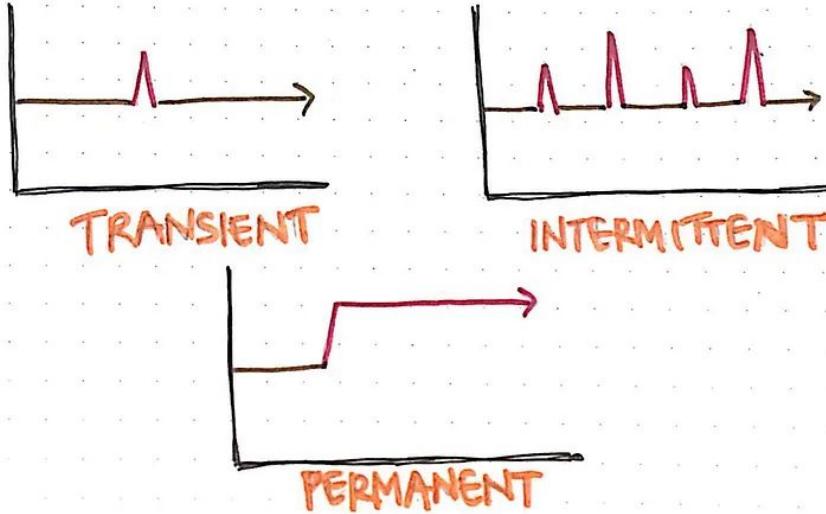
Types of Faults:



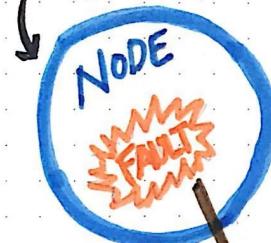
universität
wien

SBA
Research

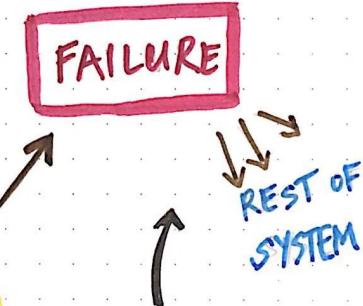
Types of Faults [1]:



The fault originates somewhere, and is a flaw in the system.



When the fault causes unexpected behavior, we get an unexpected/incorrect result: an **ERROR**.



If the error is not handled/hidden, the node will return an incorrect result, and result in a **FAILURE**.

[1] <https://medium.com/baseds/fantastic-faults-and-what-to-call-them-56d91a1b198c>

Types of Failures that can be caused by Faults:



universität
wien

SBA
Research

Types of Failures that can be caused by Faults:

- timing failure
 - E.g. Messages arrive too early or too late, processing takes longer than expected
- omission failure
 - E.g. Message is lost in transit
- crash failure
 - E.g. computer goes up in flames
- arbitrary failure
 - E.g. Message is corrupted, any other type of failure etc.
- Byzantine failure
 - Different view on arbitrary failures where failure is caused by an attacker

Note: This list is not exhaustive! These are the most common types of failures that are addressed



Byzantine Failure:

- Arbitrary failure model viewed under the following perspective:
A sequence of arbitrary failures, in the worst case, could behave exactly the same as an adversary trying to attack the system



Note: The term "Byzantine" stems from a seminal paper on consensus by Lamport et al.[1]

[1] <https://allquantor.at/blockchainbib/#lamport1982byzantine>

Difficulty: Communication Delays (and Timing Failures)



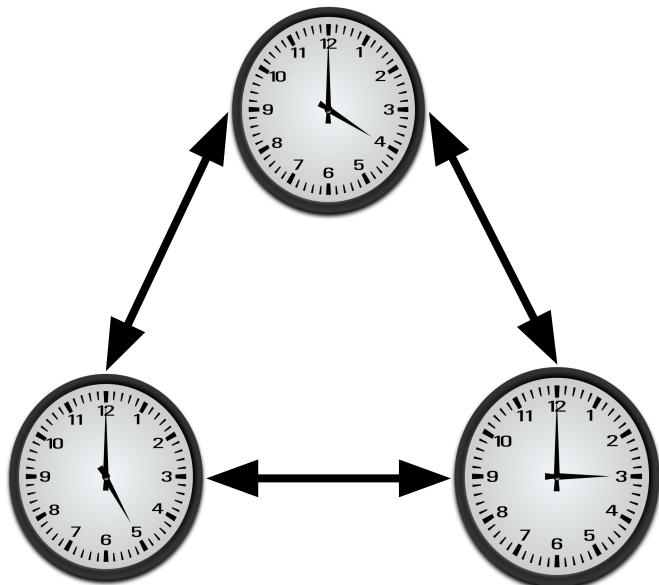
Difficulty: Communication Delays (+ Timing Failures)

- Generally speaking, nodes within a distributed system have individual clocks, so there is **no global time**
- Lack of a global clock means the **timing of events**, and hence their temporal order, **can be subjective** for each node, i.e., there is **no (implicit) agreement on time**
- Difficult to discern e.g., between communication delay of a message on the network or a faulty/inaccurate local clock

How can we address issues with timing?



How can we address issues with timing?

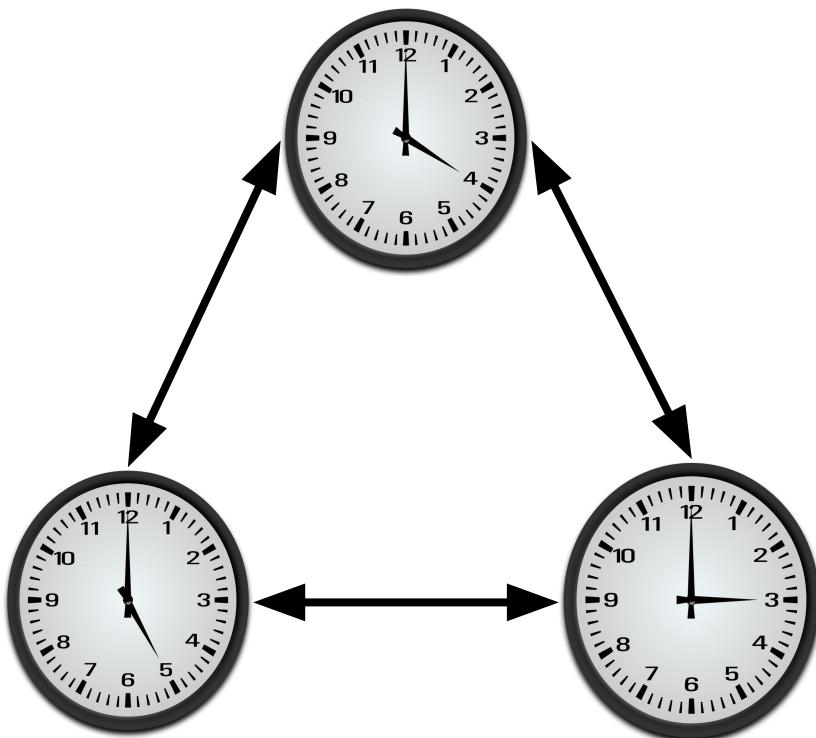


Improve accuracy of clocks and perform clock synchronization



Try to design protocols that rely as little as possible on concrete timing, i.e. are mostly asynchronous

Clock Synchronization



Clock Synchronization using Cristian's Algorithm[1]

Basic Idea:

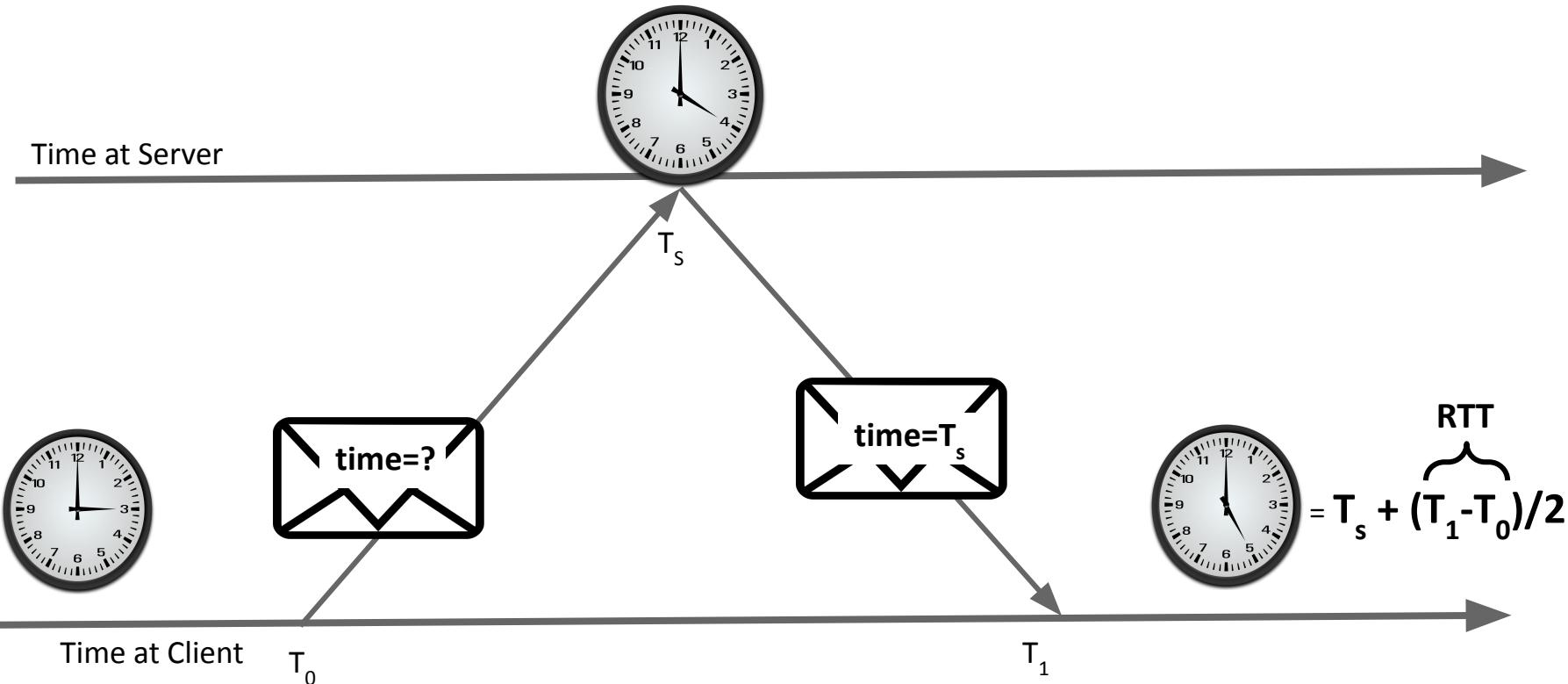
- Client server model
- Time server with accurate source of time
- Client sends a request to the server asking for the time and notes down its local time (T_0) when it sent the message
- The server responds with its local time (T_s) when it received the message
- Once the client receives the response from the server, it notes down the arrival time (T_1), computes half the delay from the Round Trip Time (RTT) i.e., $(T_1 - T_0)/2$, adds this value to T_s and adjusts its local clock to the result

Client clock deviance from Server after synchronization is at most $(T_1 - T_0)/2$

If the message delay in both directions is the same, synchronization is error-free

[1] Cristian, F. Probabilistic clock synchronization. *Distrib Comput* 3, 146–158 (1989).

Clock Synchronization using Cristian's Algorithm[1]



[1] Cristian, F. Probabilistic clock synchronization. *Distrib Comput* 3, 146–158 (1989).

Potential Issues with Christian's Algorithm?



Potential Issues with Cristian's Algorithm?

- Assumes communication delay is roughly equal in both directions
- Generally relies on low latency network (for accuracy)
- Requires trusted time server
- Time may run backward
- RTT of subsequent requests can experience (potentially large) variation
- No redundancy



Logical Clocks



Source: <https://www.youtube.com/watch?v=u8ccGjar4Es>



universität
wien

SBA
Research

Logical Clocks

Basic Idea of a logical clock (e.g. Lamport Timestamp):

- We are often not interested in the concrete time an event happened, but rather when events happened in relation to each other (e.g. **a** happened before **b** (written as $a \sim b$))
- Instead of measuring absolute time and time differences, a process tracks the occurrence of events with a counter (timestamp), which is incremented whenever an event happens. (The counter is always incremented, i.e., time can only move forward!)
- We say that some event **a** happened before **b** at some process if the counter/timestamp for **a** is smaller than **b**, e.g. $C(a) < C(b)$
- A process updates (increments) and attaches its counter/timestamp when sending messages, and upon receiving messages a process uses the attached counter to update/synchronize by setting its own counter to the maximum of either counter values and increasing it by one



Lamport Timestamps

"To synchronize logical clocks, Lamport defined a relation called happens-before. The expression $a \sim b$ is read "a happens before b" and means that all processes agree that first event a occurs, then afterward, event b occurs. The happens-before relation can be observed directly in two situations:

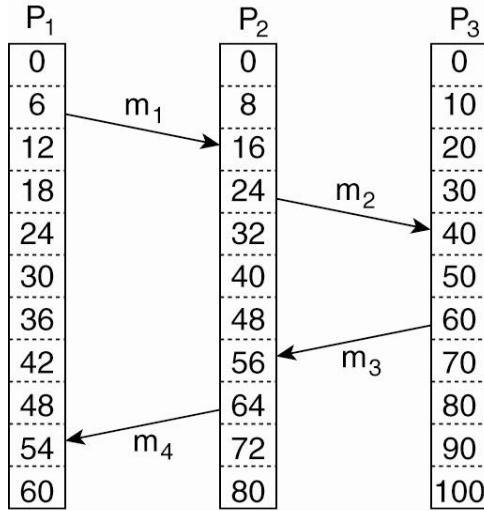
1. If a and b are events in the same process, and a occurs before b, then $a \sim b$ is true.
2. If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then $a \sim b$ is also true. A message cannot be received before or at the time it was sent.

Happens-before is a transitive relation, so if $a \sim b$ and $b \sim c$, then $a \sim c$.

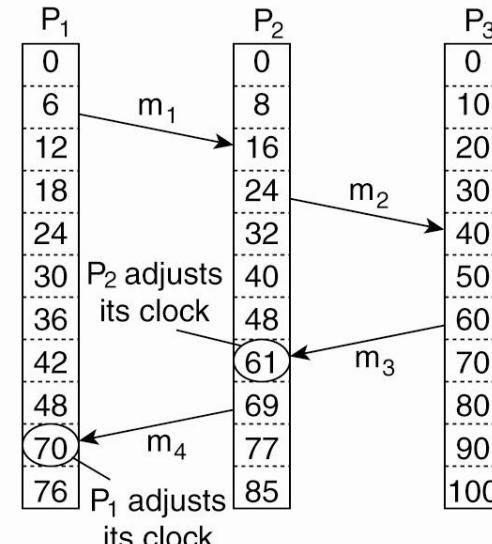
If two events, x and y, happen in different processes that do not exchange messages (not even indirectly via third parties), then $x \sim y$ is not true, but neither is $y \sim x$. These events are said to be concurrent, which simply means that nothing can be said (or need be said) about when the events happened or which event happened first."

Lamport Timestamps

•



(a)

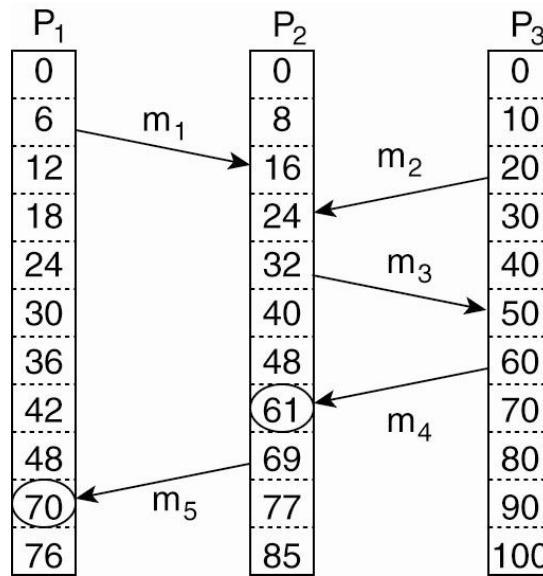


(b)

(a) Three processes, each with its own clock.
The clocks run at different rates.

(b) Lamport's algorithm corrects the clocks

Lamport Timestamps != Causal Order



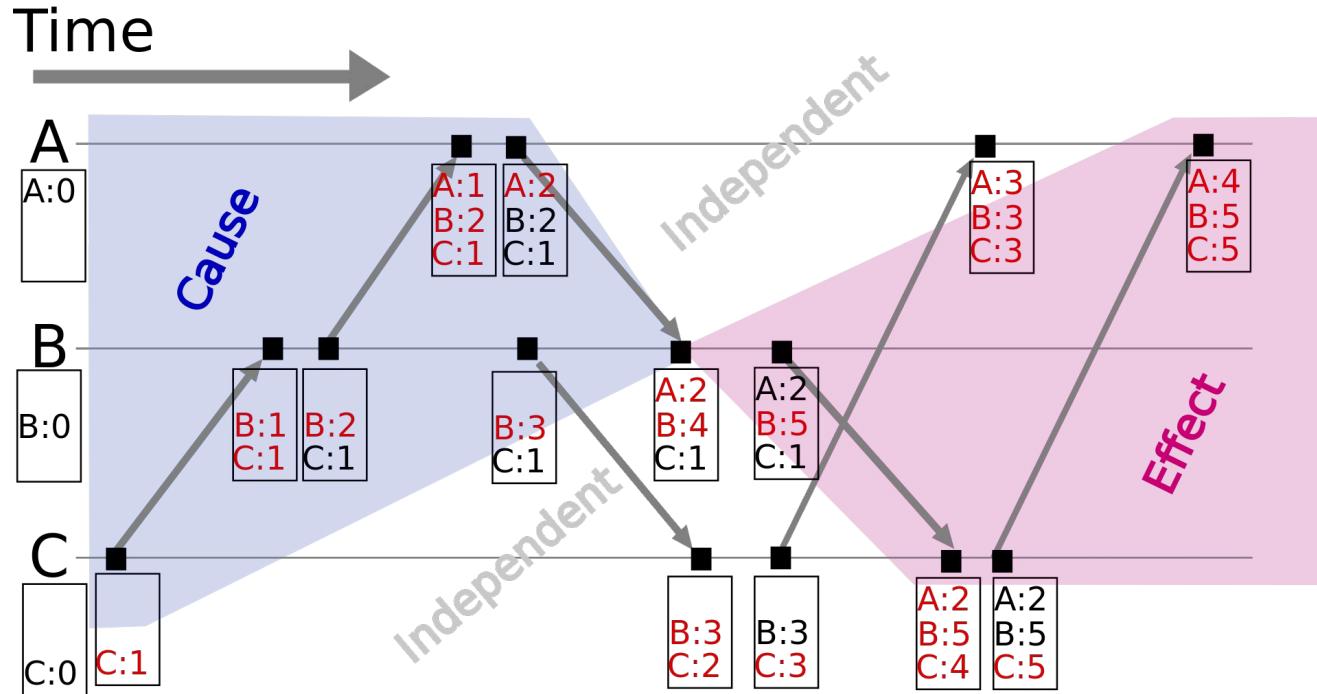
While m₁ may have been received before m₂, the sending of m₂ has nothing to do with m₁ (it is not causally dependent). This is not fully captured by Lamport timestamps

Vector clock

Basic Idea:

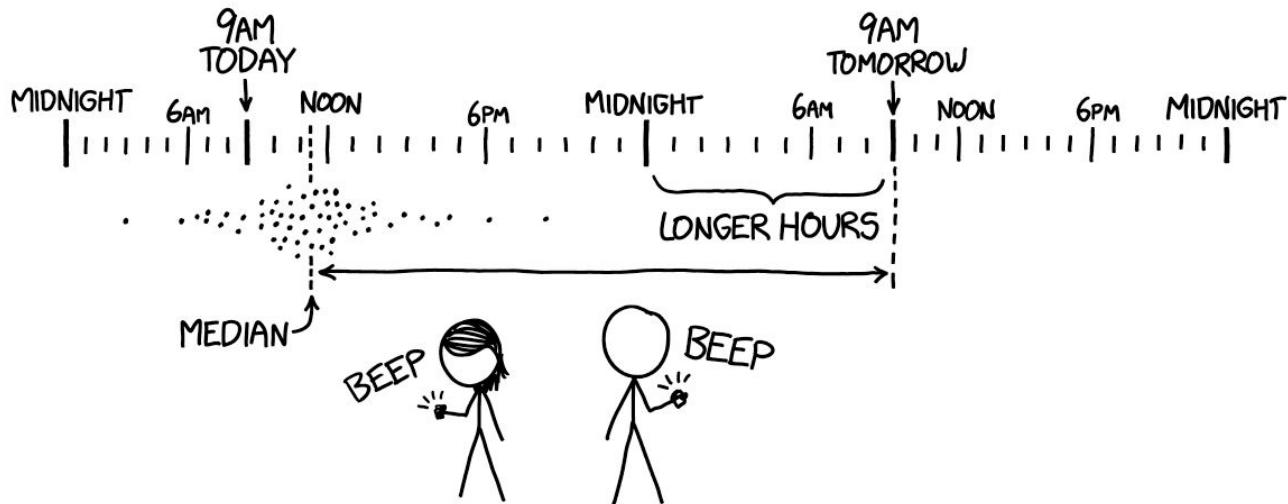
- Instead of a single combined timestamp / counter we keep a vector of counters, one for each process in the system (representing their logical clock)
- Vector clocks provide a finer granularity which allows us to obtain a causal ordering of events
- Used e.g, to implement causal order broadcast

Vector clock



PROPOSAL: CONSENSUS TIME

EVERY DAY, ANYONE IN THE TIME ZONE CAN PRESS A BUTTON WHEN THEY FEEL LIKE IT'S 9 AM. THE NEXT DAY, CLOCKS SLOW DOWN OR SPEED UP TO MATCH THE MEDIAN CHOICE FROM THE PREVIOUS DAY.



Summary of Discussed Difficulties in Distributed Computing

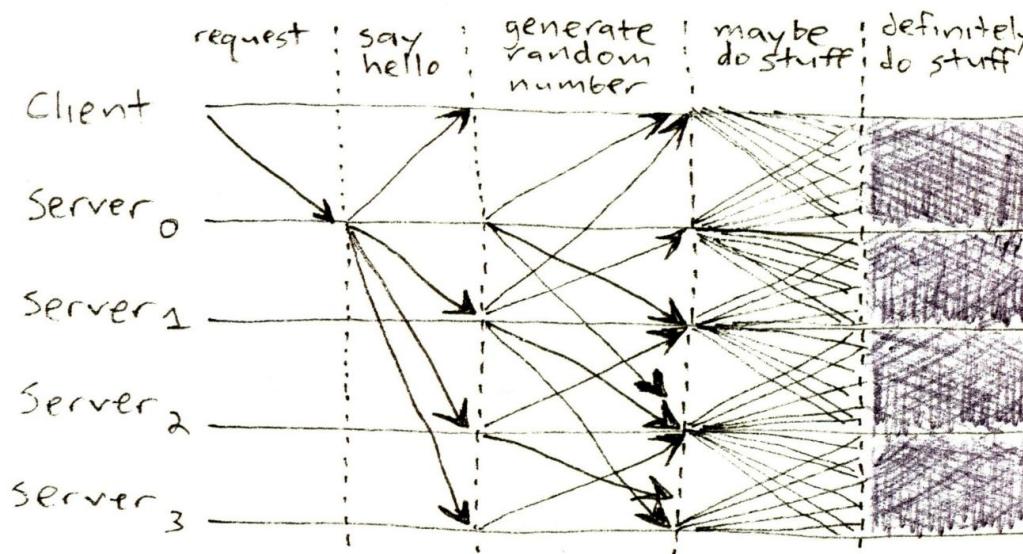
- Random faults can occur in different components
- There is no global clock that we can rely on

Especially in combination these difficulties can become hard problems to address:

- Has node i crashed or is i just slow?
(how long should we wait for a response / can we ensure liveness?)
- Can I trust the response from node j?
(how can we ensure correctness, esp. for Byzantine failures?)
- All nodes should either perform update x at time t, or none at all
(how can we ensure consistency?)

Why is it important to understand these difficulties and how problems are addressed?

- Often protocols can have weaknesses that relate to these difficulties
- Breaking the system model assumptions may mean guarantees no longer hold
- Protocols may claim to solve problems that are thought to be impossible



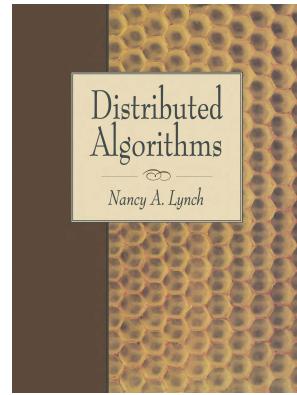
How do we approach Problems in Distributed Computing?

"The general style of work in this field is more or less as follows.

First, problems of significance in practical distributed computing are identified, and abstract versions of these problems, suitable for mathematical study, are defined.

Then, algorithms that solve the problems are developed. These are described precisely and proved to solve the stated problems, and their complexity, according to various measures, is analyzed. Designers of such algorithms typically try to minimize their complexity. Also, impossibility results and lower bounds are proved, demonstrating limitations on what problems can be solved and with what costs.

Underlying all of this work are mathematical models for distributed systems."



Approaching Problems in Distributed Computing:

1. Identify and Describe Underlying Problem
2. Define the System Model and Assumptions
3. Attempt to Solve Problem
 - a. Try to reduce to other known problem
 - b. Identify (im)possibility of solving problem
 - i. Identify bounds and limitations
 - c. Propose Algorithm that solves problem
 - i. Analyze Algorithm (e.g. complexity)



Example Problem in Distributed Computing:

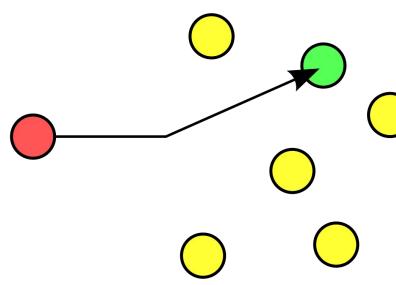
Reliable Broadcast



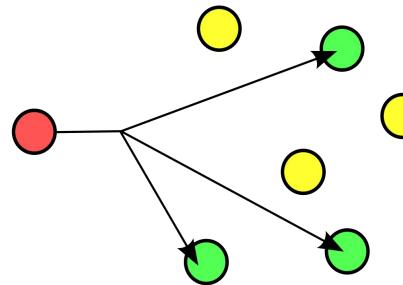
Reliable Broadcast

Basic Idea:

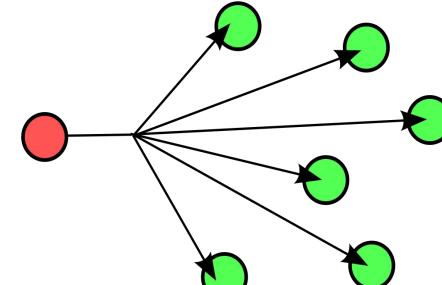
- Implement a broadcast primitive as a protocol / building block (generally using direct communication links between nodes)
- Recipients should either all receive the same broadcast message or nothing at all (all or nothing)
- Not to be confused with IP Broadcast (though the below Illustrations come from this area ;-))
- Protocol should be able to deal with faults (generally we focus on faulty nodes)



Unicast



Multicast

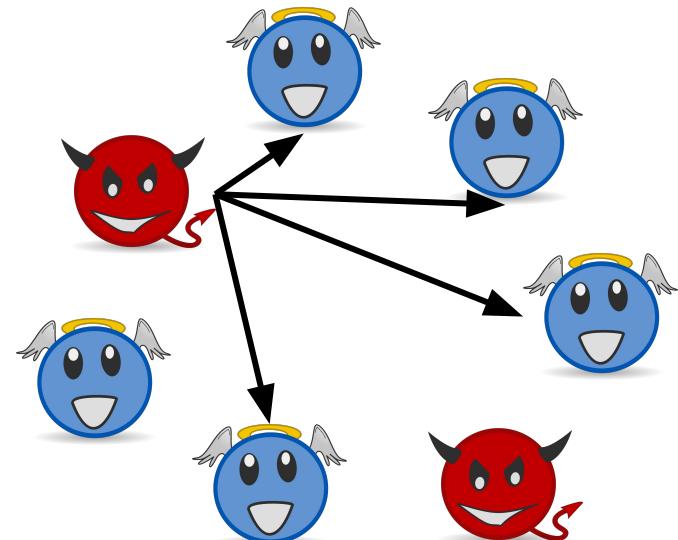


Broadcast

Reliable Broadcast Problem (Informal)

We want to implement a broadcast protocol where:

- There is a fixed set of Nodes n
- If a correct node sends a message, it will be eventually delivered by all other correct nodes (**Validity**)
- If a correct node delivers a message, then all correct nodes will eventually deliver the same message (**Agreement**)
- Arbitrary (Byzantine) failures of some subset of nodes is tolerated
- The protocol should not rely on time (**Asynchronous**)



Properties of Bracha (Reliable) Broadcast [1] (Problem)

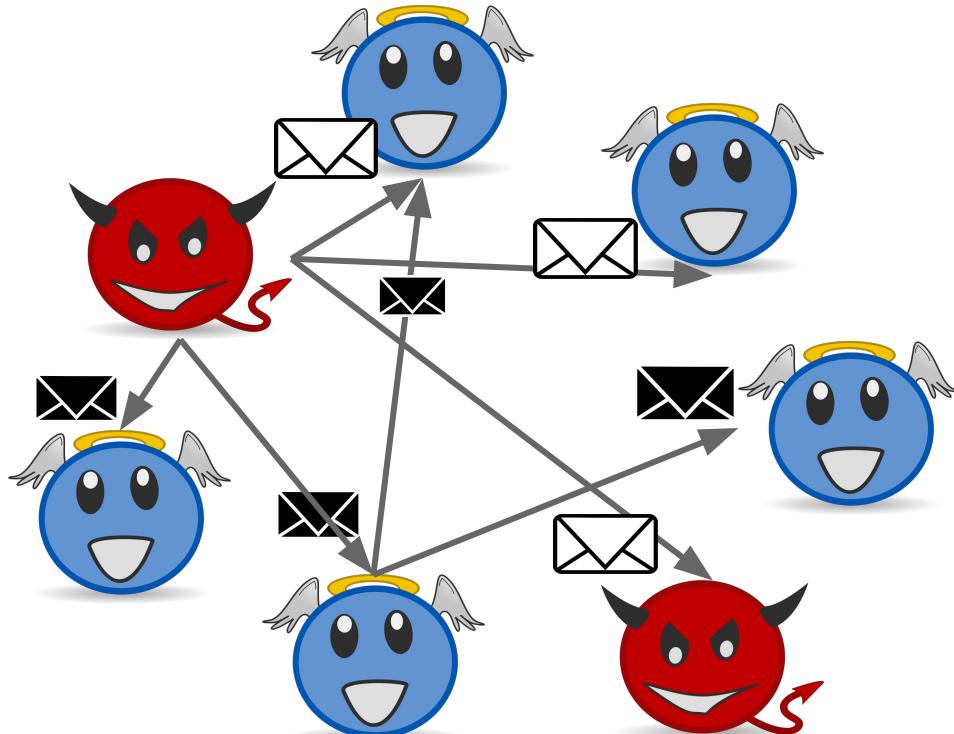
We first present a broadcast protocol that will be used as a primitive in the consensus protocol. In a single instance of the broadcast protocol, some designated process, p , sends messages containing its value to all other processes. The protocol is a *reliable broadcast protocol* if it satisfies the following properties:

1. if p is correct, then all correct processes agree on the value of its message;
2. if p is faulty then, either all correct processes agree on the same value or none of them accepts any value from p .

Assumed System Model for Bracha Broadcast [1]

- Asynchronous communication and processing
- Fully connected network of n nodes with bidirectional communication links
- Reliable messaging system (no messages are lost or additionally generated)
- No authenticated messages
- Processes may fail arbitrarily i.e. Byzantine failures
 - ◆ At most f of the $3f+1 = n$ processes may fail

What Solution would You Propose?



Different
 messages

Basic idea behind Bracha Broadcast:

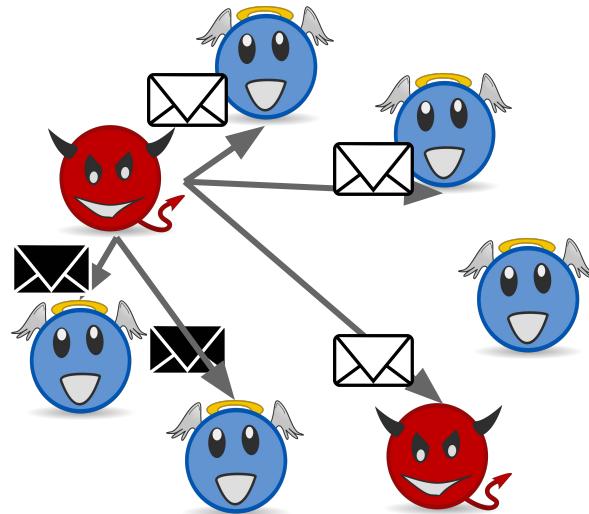
Sender(p_s), Nodes ($\{p_1 \dots p_n\}$), Message (m)

1. Sender (p_s) sends (initial, m) message to every process p_i
2. If p_i hears about m for the first time, send (echo, m) to every process p_i
3. If some p_i has heard the echo of m from "enough" other processes, send (ready, m) message to every process p_i
4. If p_i receives "enough" ready messages, it delivers the m to the application

- Basic Idea behind Bracha (Reliable) Broadcast:

Question: How much confirmation is "enough" ?

- For echo messages?
- For ready messages?



- Basic Idea behind Bracha (Reliable) Broadcast:

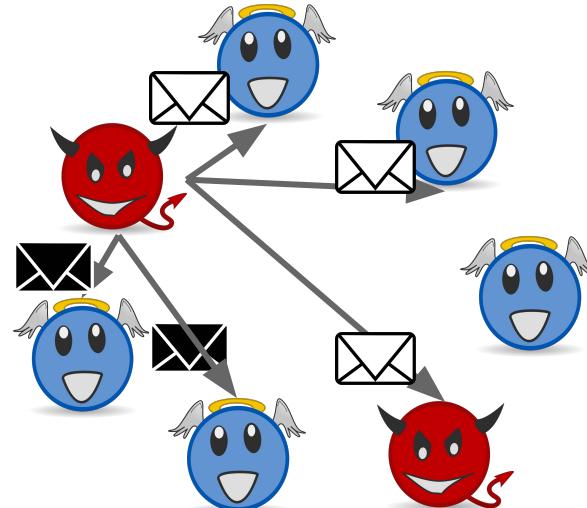
Question: How much confirmation is "enough" ?

- For echo messages?
- For ready messages?

Answer:

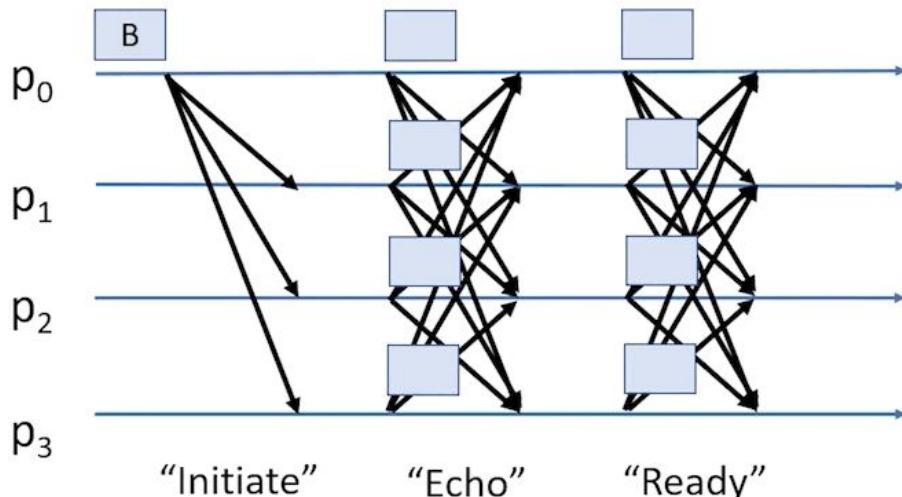
$2f+1$ echo messages

$f+1$ ready messages



Reliable Broadcast

[Bracha, *Information and Computation* 1987]



$Broadcast(v)$

step 0. (By process p)

Send $(initial, v)$ to all the processes.

step 1. Wait until the receipt of,
one $(initial, v)$ message
or $(n+t)/2 (echo, v)$ messages
or $(t+1) (ready, v)$ messages
for some v .

Send $(echo, v)$ to all the processes.

step 2. Wait until the receipt of,
 $(n+t)/2 (echo, v)$ messages
or $t+1 (ready, v)$ messages
(including messages received in step 1)
for some v .

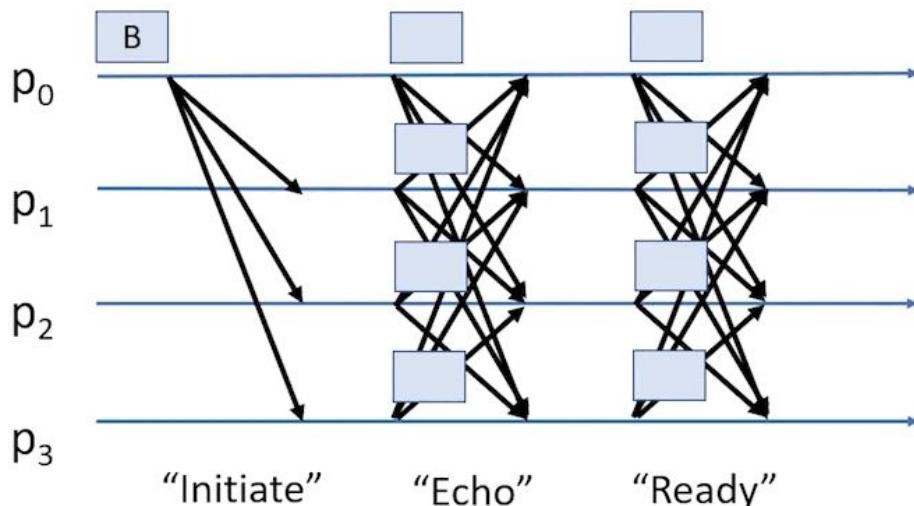
Send $(ready, v)$ to all the processes.

step 3. Wait until the receipt of,
 $2t+1 (ready, v)$ messages
(including messages received in step 1 and step 2) for some v .
Accept v .

Q: What is the communication Complexity?

Reliable Broadcast

[Bracha, *Information and Computation* 1987]



$\text{Broadcast}(v)$

step 0. (By process p)

Send (initial, v) to all the processes.

step 1. Wait until the receipt of,
one $(\text{initial}, v)$ message
or $(n+t)/2 (\text{echo}, v)$ messages
or $(t+1) (\text{ready}, v)$ messages
for some v .

Send (echo, v) to all the processes.

step 2. Wait until the receipt of,
 $(n+t)/2 (\text{echo}, v)$ messages
or $t+1 (\text{ready}, v)$ messages
(including messages received in step 1)
for some v .

Send (ready, v) to all the processes.

step 3. Wait until the receipt of,
 $2t+1 (\text{ready}, v)$ messages
(including messages received in step 1 and step 2) for some v .
Accept v.

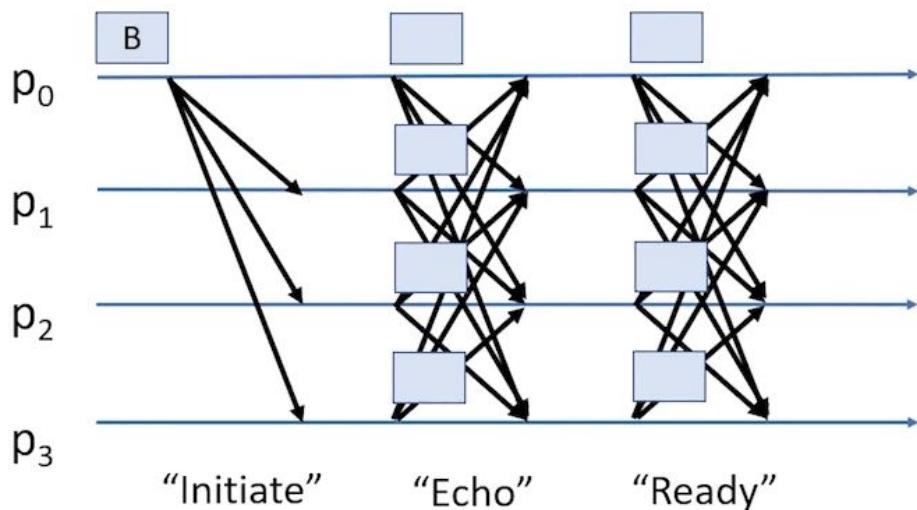
Q: What is the communication Complexity?

Reliable Broadcast

[Bracha, *Information and Computation*, 1987]

A: $O(n^2 |B|)$

processes.



step 1. Wait until the receipt of,
one $(initial, v)$ message
or $(n+t)/2 (echo, v)$ messages
or $(t+1) (ready, v)$ messages
for some v .
Send $(echo, v)$ to all the processes.

step 2. Wait until the receipt of,
 $(n+t)/2 (echo, v)$ messages
or $t+1 (ready, v)$ messages
(including messages received in step 1)
for some v .
Send $(ready, v)$ to all the processes.

step 3. Wait until the receipt of,
 $2t+1 (ready, v)$ messages
(including messages received in step 1 and step 2) for some v .
Accept v .



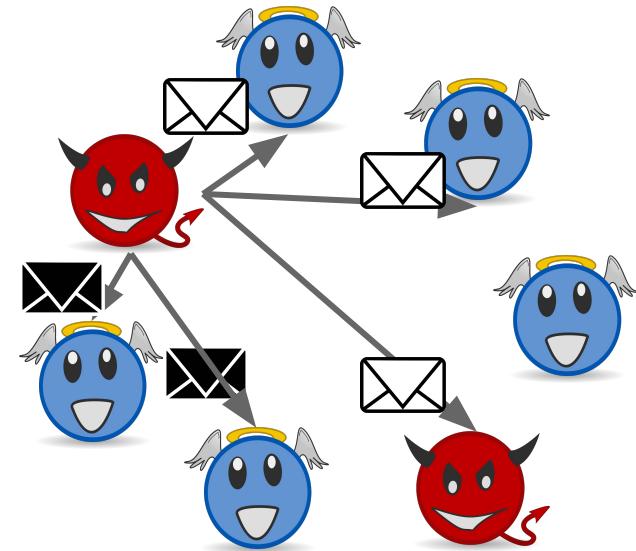
universität
wien

SBA
Research

Bracha (Reliable) Broadcast:

Some "Real World" issues:

- What happens to messages that can't be delivered? (Bounded Buffer)
- What happens if the network is really asynchronous? (e.g. Fixing the internet cables takes a couple of weeks)
- What if the number of Nodes n is big or we want to change n ?



The Consensus Problem in Distributed Computing



https://docs.google.com/presentation/d/1lsWdi0opVLj1qyDu_SxR_ZmHE6JyZ5-iUUGWwUNqZpM/edit?usp=sharing



universität
wien



Deep Dive into Distributed Consensus



universität
wien

SBA
Research

The Consensus Problem

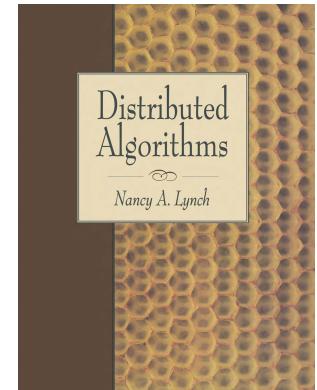
"The problem of reaching agreement among remote processes is one of the most fundamental problems in distributed computing and is at the core of many algorithms for distributed data processing, distributed file management, and fault-tolerant distributed applications." [1]

[1] Impossibility of distributed consensus with one faulty process, Michael J Fischer, Nancy A Lynch, Michael S Paterson, In Proc. of: Journal of the ACM (JACM), 1985, ACM

Refresher:

How to Approach Problems in Distributed Computing:

1. Identify and Describe Underlying Problem
2. Define the System Model and Assumptions
3. Attempt to Solve Problem
 - a. Try to reduce to other known problem
 - b. Identify (im)possibility of solving problem
 - i. Identify bounds and limitations
 - c. Propose Algorithm that solves problem
 - i. Analyze Algorithm (e.g. complexity)



The Consensus Problem (Informal)

- We have a set of processes communicating via message passing
- They want to agree on some non-trivial input
- Processes should agree on the same thing
- The protocol should eventually produce some result
- Things can fail



The Consensus Problem

Consensus Problems are generally defined through 3* properties:

- 1) **Validity:** If a process decides a value v , then v was proposed by some process
- 2) **Agreement:** No two correct processes decide differently
- 3) **Termination:** Every correct process (eventually) decides some value

- *Validity* and *Agreement* properties are referred to as **safety** properties
- The *Termination* property is there to ensure **liveness**

*some problem definitions like atomic broadcast use 4 or more

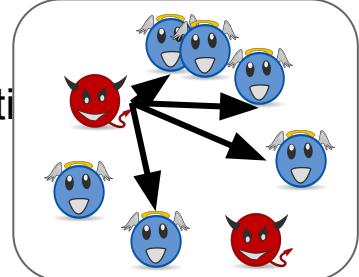
Recall the Reliable Broadcast Problem

Consensus Problems are generally defined through 3* properties:

- 1) **Validity:** If a process decides a value v , then v was proposed by some process
- 2) **Agreement:** No two correct processes decide different values
- 3) **Termination:** Every correct process (eventually) decides some value

This is missing in
Reliable Broadcast

- *Validity* and *Agreement* properties are referred to as **safety** properties
- The *Termination* property is there to ensure **liveness**



*some problem definitions like atomic broadcast use 4 or more

General System Model Assumptions for Consensus

- Total number of processes (nodes): n
- Total number of faulty nodes: f
 - sometimes also denoted with: t
- Generally a fixed Set of processes: $P = \{p_1, p_2, \dots, p_n\}$
- Usually bidirectional communication links between all nodes, i.e., fully connected communication graph
 - Often: communication links are reliable, i.e., no messages are lost, changed, created or duplicated
- Generally a fixed set of possible values to propose from
 - Often Binary Consensus is considered when describing protocols
- Disallow trivial solutions (e.g. always decide x)
 - Generally, process start with a random preference from the set of allowed votes

General System Model Assumptions for Consensus

- Total number of processes (nodes): n
- Total number of faulty nodes: f
 - sometimes also denoted with: t
- Generally a fixed Set of processes: $P = \{p_1, p_2, \dots, p_n\}$
- Usually bidirectional communication links between all nodes, i.e., fully connected communication graph
 - Often: communication links are reliable, i.e., no messages are lost, changed, created or duplicated
- Generally a fixed set of possible values to propose from
 - Often Binary Consensus is considered when describing protocols
- Disallow trivial solutions (e.g. always decide x)
 - Generally, process start with a random preference from the set of allowed votes

General System Model Assumptions (Time)

Synchrony Assumptions

Harder Problem*

- **Synchronous setting:**
 - . There is a (fixed) known upper time bound Δ that always holds
- **Partially synchronous setting(s):**
 - . Known upper bound Δ that eventually holds
 - At some Global Stabilization Time (GST)
 - . Unknown upper bound Δ that always holds
 - . Unknown upper bound Δ that eventually holds
- **Asynchronous setting:**
 - . No upper bound
 - If communication channel is reliable all messages eventually arrive

Strength of
System Model

*Generally speaking

Origins of Consensus Research

- Pease et al., *Reaching Agreement in the Presence of Faults*, 1980 [1]
- Lamport et al., *The Byzantine Generals Problem*, 1982 [2]
- Fischer et al., *Impossibility of Distributed Consensus with One Faulty Process*, 1985 [3]
- Michael Rabin, *Randomized Byzantine Generals*, 1983 [4]
- Michael Ben-Or, *Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols*, 1983 [5]
- Dwork et al., *Consensus in the presence of partial synchrony*, 1988 [6]

Impossibility of Consensus with one Faulty Process (FLP Impossibility)



Impossibility Distributed Consensus with One Faulty Process

Seminal Paper [1] by Fischer, Lynch and Paterson on Asynchronous Consensus which proves a fundamental impossibility result for Consensus Problems

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the “Byzantine Generals” problem.

[1] Impossibility of distributed consensus with one faulty process, Michael J Fischer, Nancy A Lynch, Michael S Paterson, In Proc. of: Journal of the ACM (JACM), 1985, ACM

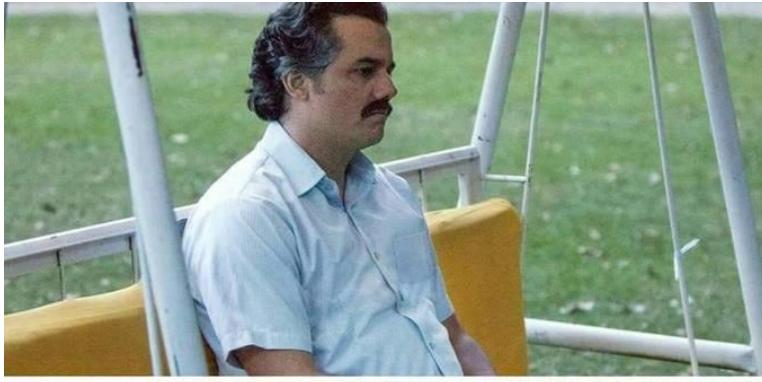
Impossibility of Consensus with one Faulty Process

System Model and Assumptions:

- **Asynchronous** System Model
- $n \geq 2$ processes
- Reliable Broadcast primitive available*
 - Communication links are reliable, i.e. no messages lost
- Processes and communication may be arbitrarily slow
- Processes may crash (crash-failures / fail-stop)
- Goal: Binary Consensus (agree either on 0 or 1)

*the paper calls this atomic broadcast, however this term is nowadays generally used to describe a different consensus problem.

Informal Intuition behind the FLP Impossibility Result:



+



Image Sources: <https://i.kym-cdn.com/photos/images/original/001/373/328/b16.jpg>
<https://imgflip.com/memegenerator/Two-Buttons>



universität
wien

SBA
Research

Informal Intuition behind the FLP Impossibility Result:

The consensus protocol needs to ensure that the agreement property holds. At the same time it needs to ensure termination.

- If even a single process can crash, there is no deterministic rule where we can terminate the protocol such that for all possible executions the processes will decide, i.e., agree on the same value
 - Basically, in some unlucky executions where processes may fail halfway through the protocol we have to either wait (indefinitely) for their answers or risk some processes deciding differently

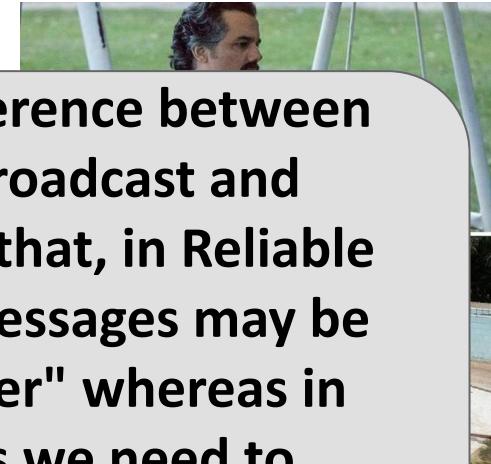


Informal Intuition behind the FLP Impossibility Result:

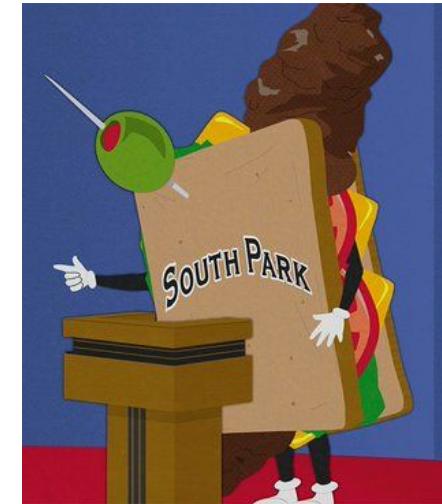
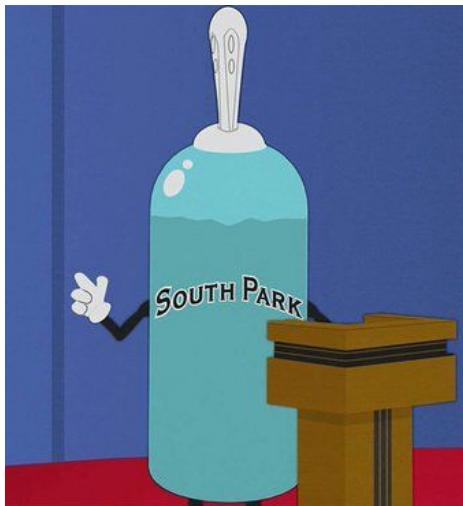
The consensus protocol needs to ensure that the agreement property holds. At the same time it needs to ensure termination.

- If even a single process can terminate, then there must be a point in time where we can tell which value each process will decide on. This means that for all possible executions the processes must eventually decide, i.e., agree on the same value.
 - the "crashed" process could just be very slow
 - If we are right at the edge between choosing 0 or 1, a slow process may end up deciding differently.

The core difference between Reliable Broadcast and Consensus is that, in Reliable Broadcast, messages may be "stuck forever" whereas in Consensus we need to eventually agree



Informal Intuition behind the FLP Impossibility Result:

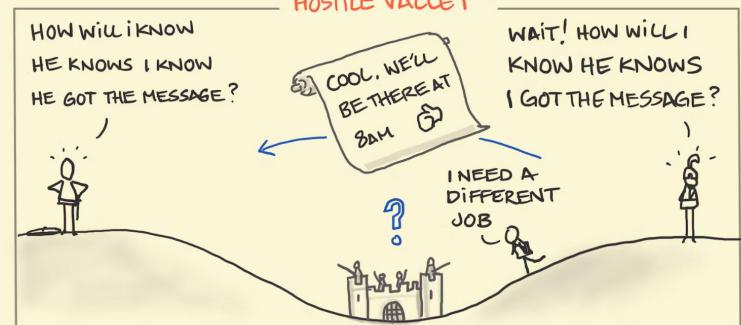
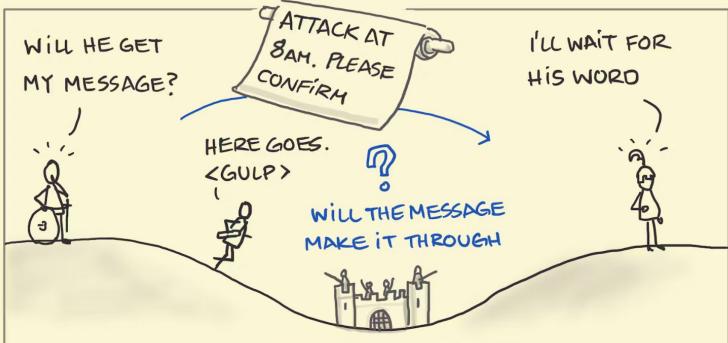


Example: if processes are roughly split between two votes, in an asynchronous system we can get "stuck" trying to resolve what to decide. In the worst case we either keep flip-flopping forever or risk having some participants decide differently

THE TWO GENERALS PROBLEM

CHALLENGES OF COMMUNICATING ON AN UNRELIABLE CHANNEL

TWO ARMIES SEPARATED BY A HOSTILE VALLEY NEED TO COORDINATE THEIR ATTACK TO WIN. BUT THEIR MESSENGERS MAY NOT MAKE IT THROUGH WITH THEIR MESSAGES.



Source: <https://sketchplanations.com/the-two-generals-problem>

FLP as a generalization of the Two Generals Problem





Michael Ben-Or &
Michael O. Rabin

Source: <https://imgflip.com/memegenerator/The-Scroll-Of-Truth>

*Deterministic

Approaches to Overcome FLP for Asynchronous Consensus

Modify (weaken) the problem slightly:

- a. Reach agreement with some probability of error
- b. Terminate with probability 1
- c. Only have a partial solution for some configurations [1]:

THEOREM 2. *There is a partially correct consensus protocol in which all non-faulty processes always reach a decision, provided no processes die during its execution and a strict majority of the processes are alive initially.*

[1] Impossibility of distributed consensus with one faulty process, Michael J Fischer, Nancy A Lynch, Michael S Paterson, In Proc. of: Journal of the ACM (JACM), 1985, ACM

Randomized Consensus

Randomization can be used to overcome the FLP impossibility result. The concept of randomized consensus algorithms was pioneered by Michael Ben-Or [1] and Michael O. Rabin [2], who jointly received the Dijkstra Prize in Distributed Computing in 2015 for their seminal work in this area.

[1] Michael Ben-Or for “Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols” in Proceedings of the Second ACM Symposium on Principles of Distributed Computing, pages 27-30, August 1983.

[2] Michael O. Rabin for “Randomized Byzantine Generals” in Proceedings of Twenty-Fourth IEEE Annual Symposium on Foundations of Computer Science, pages 403-409, November 1983.]

Randomized Consensus or: using Coin Tossing to overcome FLP



Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.



Original Image Source:

<https://www.reddit.com/media?url=https%3A%2F%2Fi.redd.it%2F28jo1h8701941.jpg>

Randomized Consensus or: using Coin Tossing to overcome FLP

Intuition for randomized consensus:

- Use a random coin for selecting what to propose in those execution runs of the protocol where we may have trouble forming a decision
- Eventually, we get lucky and a sufficiently large majority adopts a proposal that allows us to safely reach a decision
- Basically the randomness allows us to "break out" of our indecision

Protocols may rely on individual coins (Ben-Or) or a common coin (Rabin). The former generally takes longer to converge, i.e. the expected number of rounds for the protocol to complete, in particular if there are many faulty processes. The latter requires a stronger system model/setup, in particular, some kind of common reference string or distributed key generation (DKG) is assumed in order to implement a shared "coin"

[1] Michael Ben-Or for "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols" in Proceedings of the Second ACM Symposium on Principles of Distributed Computing, pages 27-30, August 1983.

[2] Michael O. Rabin for "Randomized Byzantine Generals" in Proceedings of Twenty-Fourth IEEE Annual Symposium on Foundations of Computer Science, pages 403-409, November 1983.]

Types of Randomized Consensus

Las Vegas Randomized Consensus

- Guarantees correctness (agreement property always holds)
- Low constant number of expected rounds
- Termination with probability 1
- Consensus finality (decision will not change)

Monte Carlo Randomized Consensus

- Does not guarantee correctness (configurable, non-zero probability of error)
- Generally finite, fixed number of rounds
- No guaranteed consensus finality (decision may change)



Types of Randomized Consensus

Las Vegas Randomized Consensus

- Guarantees correctness (agreement property always holds)
- Low constant number of expected rounds
- Termination with probability 1
- Consensus finality (decision will not change)

The general dominant approach
by the scientific community
(before Bitcoin)

Monte Carlo Randomized Consensus

- Does not guarantee correctness (configurable, non-zero probability)
- Generally finite, fixed number of rounds
- No guaranteed consensus finality (decision may change)

Type of Consensus Bitcoin and
many Blockchain Protocols uses
(Nakamoto Consensus)

Consensus under Partial Synchrony



Consensus under Partial Synchrony

Recall:

- Asynchronous = no upper bound on how long something may take
- Synchronous = known upper bound Δ that always holds

Partially synchronous system models lie in between these two extremes. Commonly encountered variants are:

- Known upper time bound Δ that eventually holds
 - Unknown upper time bound Δ that always holds
 - Unknown upper time bound Δ that eventually holds
- GST



Global Stabilization Time (GST) When?



Source: <https://knowyourmeme.com/memes/confused-travolta>



universität
wien

SBA
Research

Global Stabilization Time (GST)

- GST is used in the context of communication synchrony
- GST in a nutshell: Assume that after some initial time where messages can be late the network eventually stabilizes and everything adheres to our upper bound Δ from that point onward

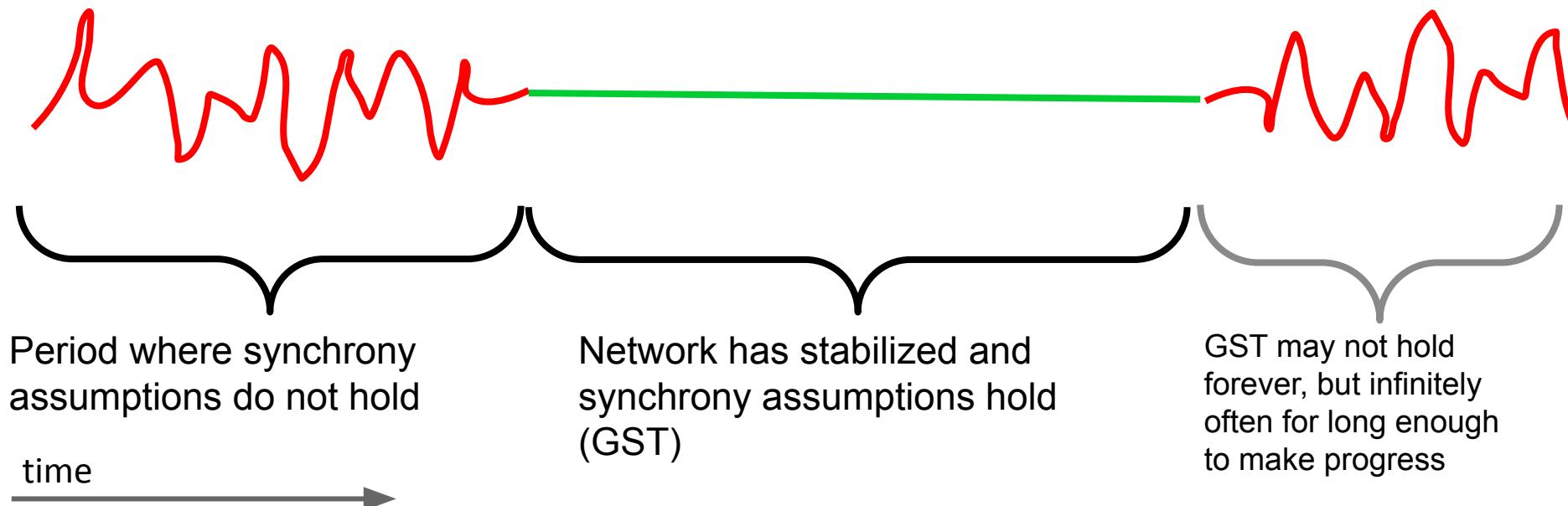
"In realistic situations, the upper bound cannot reasonably be expected to hold forever after GST, but perhaps only for a limited time. However, any good solution to the consensus problem in this model would have an upper bound L on the amount of time after GST required for consensus to be reached; in this case it is not really necessary that the bound Δ hold forever after time GST, but only up to time $\text{GST} + L$." [1]

[1] Dwork, Cynthia, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony." Journal of the ACM (JACM) 35.2 (1988): 288-323.



Global Stabilization Time (GST)

- Conceptualization of GST



Sacrifice Liveness under Periods of Asynchrony for Safety

A general approach for (consensus) protocols in models of partial synchrony is to prioritize safety over liveness. This means in the worst case while the protocol may not make progress if the network behaves too asynchronously, it will guarantee the agreement and validity (i.e. safety) properties of the protocol hold at all times.

- Intuition: basically the protocol "waits out bad network conditions" until things improve (i.e., GST holds). If GST holds infinitely long or long enough infinitely often to make at least a little progress, the protocol should eventually terminate



Synchrony Bounds for Deterministic Consensus

Consensus in the Presence of Partial Synchrony

291

TABLE I. SMALLEST NUMBER OF PROCESSORS N_{\min} FOR WHICH A t -RESILIENT
CONSENSUS PROTOCOL EXISTS

Failure type	Syn- chronous	Asyn- chronous	Partially syn- chronous com- munication and synchronous processors	Partially syn- chronous communication and pro- cessors	Partially syn- chronous pro- cessors and synchronous communica- tion
Fail-stop	t	∞	$2t + 1$	$2t + 1$	t
Omission	t	∞	$2t + 1$	$2t + 1$	$[2t, 2t + 1]$
Authenticated Byzantine	t	∞	$3t + 1$	$3t + 1$	$2t + 1$
Byzantine	$3t + 1$	∞	$3t + 1$	$3t + 1$	$3t + 1$

Solving Asynchronous* Consensus using Failure Detectors

BRYAN: CHRIS IS FAULTY.

CHRIS: CHRIS IS NOT FAULTY.

RICH: I VERIFY THAT BRYAN SAYS THAT CHRIS IS FAULTY.

BRYAN: I VERIFY MY VERIFICATION OF MY CLAIM THAT RICH CLAIMS THAT I KNOW CHRIS.

JAMES: I AM SO HUNGRY.

CHRIS: YOU ARE NOT HUNGRY.

RICH: I DECLARE CHRIS TO BE FAULTY.

CHRIS: I DECLARE RICH TO BE FAULTY.

Solving Consensus using Failure Detectors

Intuition:

- The problem of solving consensus in async. systems is that we can not decide if a process has failed or is just slow
- Pretend that there exists an oracle - a magic failure detector that tells us if a process has failed or not
- We can abstractly reason about the properties of this failure detector and how (un)reliable it can be for us to still be able to solve consensus
- We can then see if it is possible to implement the failure detector in some given system model

This approach of describing the solvability of (consensus) problems by the (weakest) Failure Detector they require was pioneered by Chandra and Toueg [1]

[1] Chandra, Tushar Deepak, and Sam Toueg. "Unreliable failure detectors for reliable distributed systems." Journal of the ACM (JACM) 43.2 (1996): 225-267.

Classes of Failure Detectors (generally for Crash-Failures)

Completeness	Accuracy			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	<i>Perfect</i> \mathcal{P}	<i>Strong</i> \mathcal{S}	<i>Eventually Perfect</i> $\diamond\mathcal{P}$	<i>Eventually Strong</i> $\diamond\mathcal{S}$
Weak	\mathcal{Q}	<i>Weak</i> \mathcal{W}	$\diamond\mathcal{Q}$	<i>Eventually Weak</i> $\diamond\mathcal{W}$

Accuracy:

- Strong -> never suspects *any* process before it fails
- Weak -> *some* correct process is never suspected of having failed

Note: There is an algorithm for transforming weak into strong completeness

Completeness:

- Strong -> eventually *every* crashed process is permanently suspected by *all* correct processes
- Weak -> eventually *every* crashed process is permanently suspected by *some* process

[1] Chandra, Tushar Deepak, and Sam Toueg. "Unreliable failure detectors for reliable distributed systems." Journal of the ACM (JACM) 43.2 (1996): 225-267.

Realizing an Eventually Perfect Failure Detector

Eventually Perfect
 $\diamond \mathcal{P}$

Intuition for building an eventually perfect failure detector (for crash-failures):

- Have a local failure detector module that tracks, for each process, if it *suspects* it of having failed (crashed)
 - Failure detector has some default time-out value defined and keeps track of when it last heard from processes.
 - Process p_i periodically sends out " p_i is-alive" messages to other processes
 - If failure detector doesn't hear from p_i before time-out it *suspects* p_i of having failed
 - If failure detector receives message from p_i while it *suspects* p_i - it *unsuspects* the process and increases the time-out value
- To transform any failure detector with eventually weak completeness into an eventually strong one, every process simply periodically disseminates the suspected processes from its failure detector to other processes. Upon receiving such a message, a process also starts to suspect all the processes suspected in the message



Realizing an Eventually Perfect Failure Detector

Eventually Perfect
 $\Diamond\mathcal{P}$



Every process p executes the following:

```
outputp ← ∅           Set of processes (i.e., n)
for all  $q \in \Pi$ 
    Δp(q) ← default time-out interval
    {Δp(q) denotes the duration of p's time-out interval for q}
```

```
cobegin
    || Task 1: repeat periodically
        send "p-is-alive" to all

    || Task 2: repeat periodically
        for all  $q \in \Pi$ 
            if  $q \notin output_p$  and
                p did not receive "q-is-alive" during the last Δp(q) ticks of p's clock
                outputp ← outputp ∪ {q}           {p times-out on q: it now suspects q has crashed}

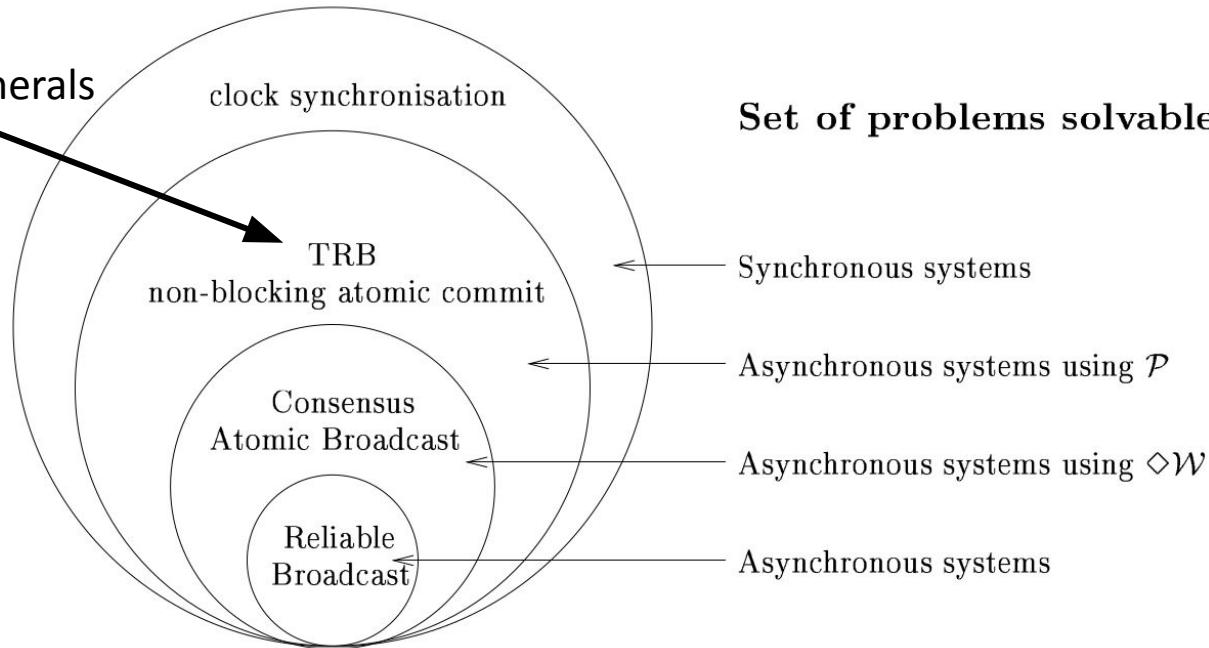
    || Task 3: when receive "q-is-alive" for some q
        if  $q \in output_p$                          {p knows that it prematurely timed-out on q}
            outputp ← outputp - {q}          {1. p repents on q, and}
            Δp(q) ← Δp(q) + 1             {2. p increases its time-out period for q}

coend
```

FIG. 10. A time-out based implementation of $\mathcal{D} \in \Diamond\mathcal{P}$ in models of partial synchrony.

Strength of Failure Detector required for Solving Problem

Byzantine Generals
Problem



*TRB = Terminating Reliable Broadcast

[1] Chandra, Tushar Deepak, and Sam Toueg. "Unreliable failure detectors for reliable distributed systems." *Journal of the ACM (JACM)* 43.2 (1996): 225-267.

The Weakest Failure Detector to Solve Consensus is in $\Diamond\mathcal{W}$

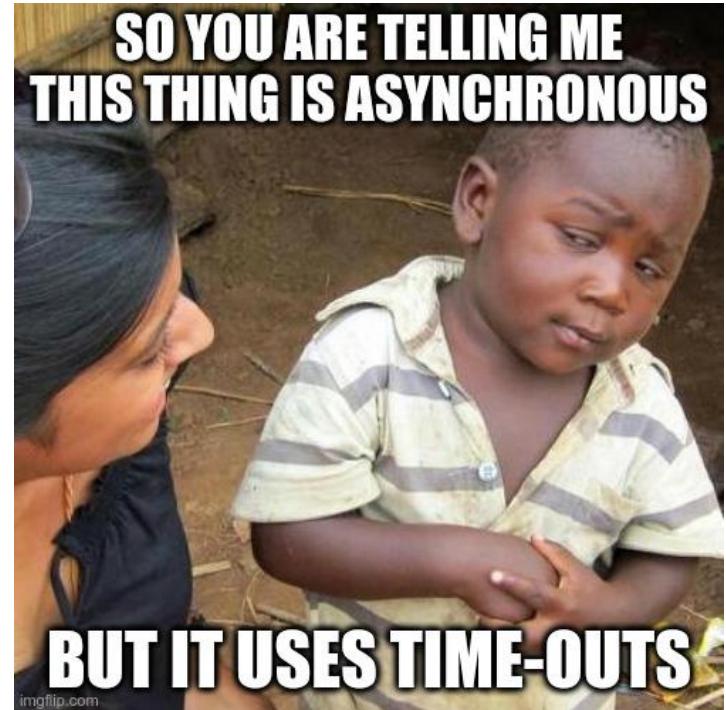
Intuition:

- In an eventually weakly accurate and weakly complete failure detector *some* correct process is eventually never suspected of having failed
- This kind of weakest failure detector for detecting such a correct process is generally referred to as an Omega (Ω) failure detector and said to provide an eventual leader election functionality
 - Eventually every process will end up not suspecting (e.g. trusting) at least one correct process
 - This can be used to drive forward the protocol to completion

The Failure Detector Abstraction can't solve the Impossible!

- Implementing the failure detector itself actually requires some form of (partial) synchrony
- It is just a (somewhat formal) way of abstracting away parts of the problem and dealing with them separately
- Arguably allows for the bulk of protocol code to be "asynchronous" as concrete timings are hidden in the failure detector implementation
- Very present model in earlier scientific publications on crash-fault tolerant consensus
- Not so common (or arguably useful) in the context of Byzantine Fault Tolerance*

*e.g. some forms of Byzantine faults are impossible to detect while others are hard to quantify (think of an adversary changing the order of events to gain a monetary advantage)



Byzantine Consensus



Origins of (Byzantine) Consensus Research

- Researchers at NASA's Langley Research Center realized that clock synchronization under an arbitrary failure model is more difficult than anticipated
- Essentially, the underlying problem requires reaching some form of *agreement* or *consensus* in a distributed setting
- The seminal papers "*Reaching agreement in the presence of faults*"[1] and the "*Byzantine generals problem*"[2] helped spark* a flurry of research in this direction.

*Among others

[1]<https://allquantor.at/blockchainbib/#pease1980reaching>

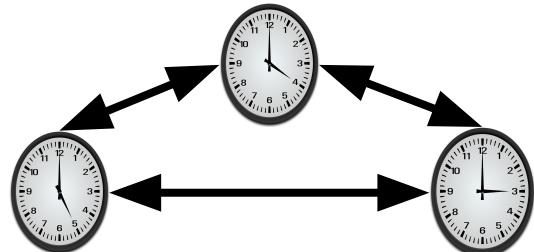
[2]<https://allquantor.at/blockchainbib/#lamport1982byzantine>



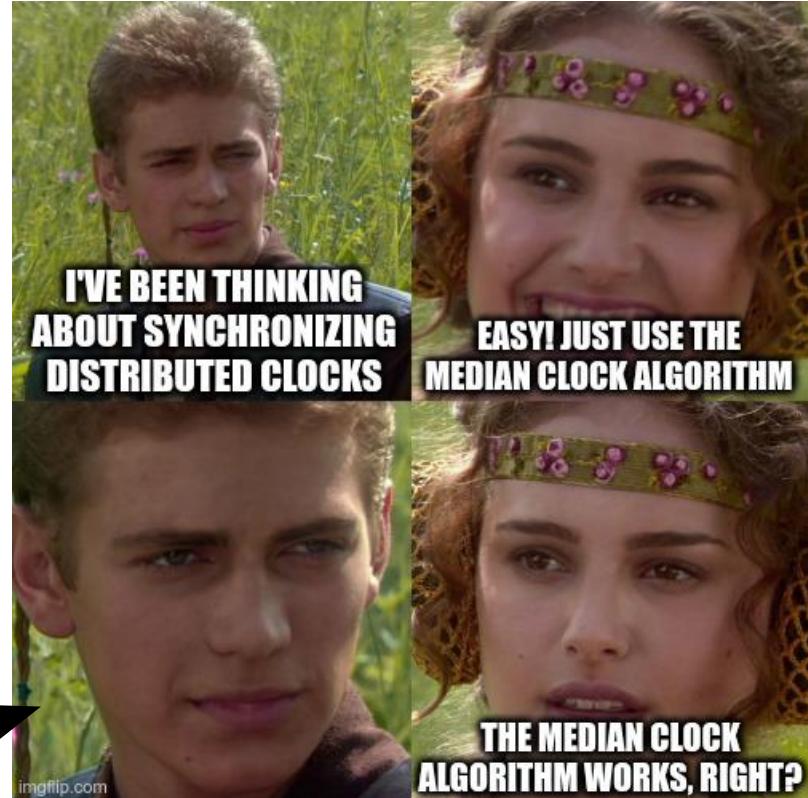
Byzantine Consensus

Recall the previous problem of clock synchronization:

- What if we do not have a dedicated time server but instead want some network of computers to synchronize their clocks among each other?
- What if clocks can fail arbitrarily?



Lamport et al.

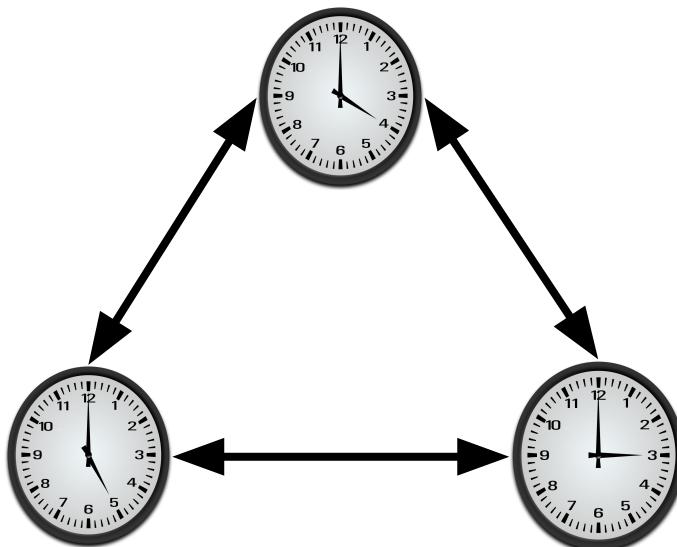


universität
wien

SBA
Research

SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control

JOHN H. WENSLEY, LESLIE LAMPORT, JACK GOLDBERG, SENIOR MEMBER, IEEE,
MILTON W. GREEN, KARL N. LEVITT, P. M. MELLiar-SMITH, ROBERT E. SHOSTAK,
AND CHARLES B. WEINSTOCK



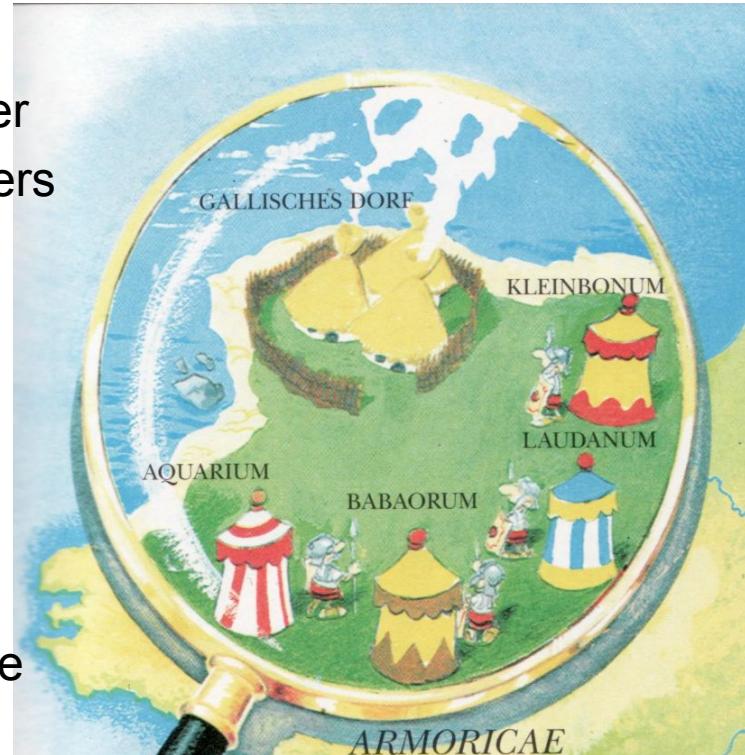
The traditional clock resynchronization algorithm for reliable systems is the median clock algorithm, requiring at least three clocks. In this algorithm, each clock observes every other clock and sets itself to the median of the values that it sees. The justification for this algorithm is that, in the presence of only a single fault, the median value must either be the value of one of the valid clocks or else it must lie between a pair of valid clock values. In either case, the median is an acceptable value for resynchronization. The weakness of this argument is that the worst possible failure modes of the clock may cause other clocks to observe different values for the failing clock. Even if the clock is read by sensing the time of a pulse waveform, the effects of a highly degraded output pulse and the inevitable slight differences between detectors can result in detection of the pulse at different times.

In the presence of a fault that results in other clocks seeing different values for the failing clock, the median resynchronization algorithm can lead to a system failure. Consider a

The Byzantine Generals Problem

- Armies need to coordinate to either attack or to retreat together
- Distinguished Leader giving orders (attack/retreat)
 - The Leader may be faulty
- Some of the armies may be disloyal (i.e., faulty)
- Communication by message passing

Goal: All honest armies should decide to do the same thing



The Byzantine Generals Problem

Problem Presented by Lamport et al. in 1982 [1]:

- Fixed set of nodes (n), one distinguished leader amongst them
- Nodes need to agree to attack or retreat
- Up to (f) nodes can be traitors (including the leader)

Paper shows that solutions require:

- $n \geq 3f + 1$ for synchronous systems with oral messages*
- $n \geq f \geq 0$ for synchronous systems with authenticated messages**
- $n \geq 3f + 1$ for asynchronous/partially synchronous systems with authenticated messages

*Oral messages: Reliable channel, receiver knows who sent message, missing messages detectable

**Authenticated messages: oral messages where additionally messages sent by correct process can't be forged and alterations can be detected, also anyone can verify the authenticity of a correct process' signature

The Byzantine Generals Problem

Byzantine Generals Problem. A commanding general must send an order to his $n - 1$ lieutenant generals such that

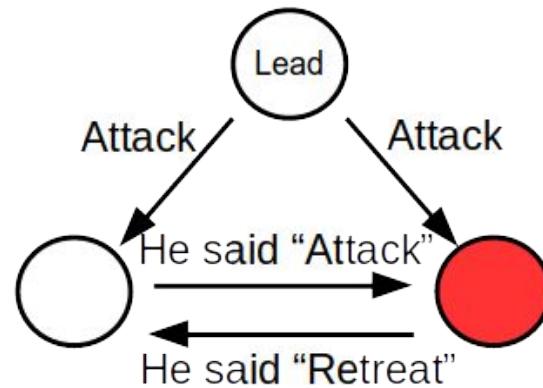
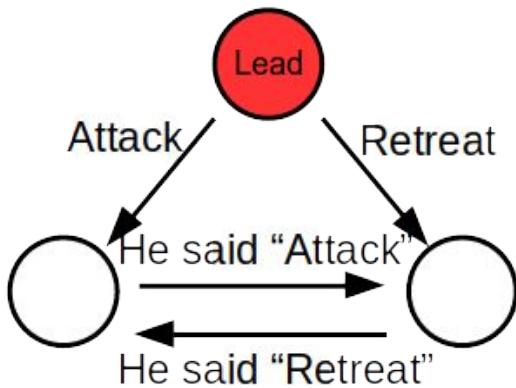
- IC1. All loyal lieutenants obey the same order.
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Conditions IC1 and IC2 are called the *interactive consistency* conditions. Note that if the commander is loyal, then IC1 follows from IC2. However, the commander need not be loyal.

[1] <https://allquantor.at/blockchainbib/#lamport1982byzantine>

The Byzantine Generals Problem

Impossibility result for $n \leq 3f^*$



- Byzantine Fault Tolerance (BFT) - generally used to refer to protocols and algorithms that are tolerant to (some threshold of) *arbitrary*, i.e., Byzantine failures
- The original solution relies on an Exponential Information Gathering (EIP) Algorithm which is not practical but shows that the problem can be solved

*depending on the system model, e.g., using oral messages or assuming partial synchrony

Further Reading on (BFT) Consensus

- Garay, Juan, and Aggelos Kiayias. "Sok: A consensus taxonomy in the blockchain era." *Topics in Cryptology–CT-RSA 2020: The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings.* Springer International Publishing, 2020.
- Zhang, Gengrui, et al. "Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms." *arXiv preprint arXiv:2204.03181* (2022).
- Wang, Gang. "SoK: Understanding BFT Consensus in the Age of Blockchains." *Cryptology ePrint Archive* (2021).



Consensus in Practice

(or: Is this at all relevant?)



Crash Fault Tolerant (CFT) Consensus in Practice

- CFT Consensus is generally used in the context of distributed databases and database replication
- Prominent examples used in practice are, for instance, Raft[1] and variants of Paxos* which can be found in:
 - Google's Chubby lock service for loosely-coupled distributed systems
 - Hyperledger Fabric ("Permissioned" Distributed Ledger)
 - Clustering e.g., in Neo4J and MongoDB



I expect that in future Byzantine fault-tolerant (BFT) consensus protocols will replace CFT

[1] <https://raft.github.io/#implementations>

*I can highly recommend reading Lamport's paper "The part-time parliament"

<https://allquantor.at/blockchainbib/#lamport1998part-time>

Byzantine Fault Tolerant Consensus in Practice

- Most Distributed Ledger Protocols use some variant of BFT Consensus
 - Bitcoin's Nakamoto Consensus (this will be covered in detail in future lectures)



- Facebook was developing LibraBFT for Libra/Diem (Based on Hot-Stuff)

Hot-Stuff the Linear, Optimal-Resilience, One-Message BFT Devil



Ittai Abraham, Guy Gueta, Dahlia Malkhi
VMware Research

March 13, 2018

Consensus Research in the Context of Security and Privacy Engineering

- Since the advent of Bitcoin and Blockchain Technologies interest in (BFT) consensus has spiked, with a plethora of new protocols being developed
 - Many of these protocols are not based on peer-reviewed, well understood concepts but developed ad-hoc (especially in the cryptocurrency space)
 - Implementations of consensus protocols can deviate (sometimes profoundly) from their specifications and formal descriptions.
- Great further reading: "Blockchain Consensus Protocols in the Wild" by Cachin and Vukolić <https://allquantor.at/blockchainbib/#cachin2017blockchain>



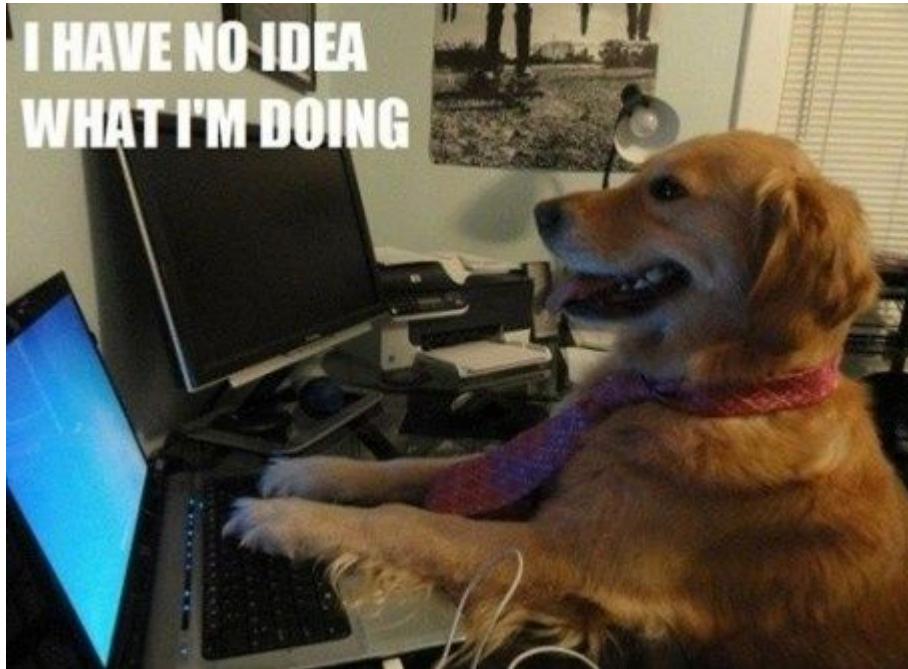
Consensus Research in the Context of SaPE

"Over the recent years countless proposals for new features in distributed ledger systems and completely new blockchain protocols have appeared. Most of them come without formal expression of their trust assumption and security model. There is no agreed consensus in the industry on which assumptions are realistic for the intended applications, not to mention any kind of accepted standard or validation for protocols.

The field of blockchain protocols is in its infancy today, but already appears at the peak of overstated expectations [4]. Many fantastic and bold claims are made in the fintech and blockchain space by startups, established companies, researchers, and self-proclaimed experts alike. Snake-oil claims appear and confuse the public opinion." [1]

[1] Cachin, Christian. "Blockchains and consensus protocols: Snake oil warning." *2017 13th European dependable computing conference (EDCC)*. IEEE, 2017.

Consensus Research in the Context of SaPE



Unfortunately, this is still a common situation - especially in the field of Cryptocurrencies

Introduction to Cryptocurrencies and Distributed Ledger Technologies (DLT)



<https://docs.google.com/presentation/d/1l4hYZb5cF7x4w5akcRxoERFePnqFvPgjj6NWiANO3Y/edit?usp=sharing>

Why should I care about Blockchain*?

* Blockchain and Distributed Ledger Technologies (DLTs)



Why should I care about Blockchain?

Coins: 13178 Exchanges: 590 Market Cap: \$971,547,754,329 0.5%↑ 24h Vol: \$48,258,439,719 Dominance: BTC 38.1% ETH 16.3% Gas: 20 GWEI

EN USD [Subscribe](#)

CoinGecko Cryptocurrencies Exchanges NFT Learn Crypto Products

Portfolio Login Sign Up Search

★ Portfolio Coins New Coins Gainers & Losers Categories NFT DeFi Gaming BNB Solana Avalanche

Cryptocurrency Prices by Market Cap Show Stats

The global cryptocurrency market cap today is \$972 Billion, a 0.5%↑ change in the last 24 hours. [Read More](#)

All Categories Show Fully Diluted Valuation

#	Coin	Price	1h	24h	7d	24h Volume	Mkt Cap	Last 7 Days
★ 1	Bitcoin BTC	\$19,297.19	0.3%	0.6%	2.5%	\$22,332,105,189	\$369,887,032,222	
★ 2	Ethereum ETH	\$1,308.43	0.8%	0.9%	1.1%	\$7,477,416,947	\$158,030,611,829	
★ 3	Tether USDT	\$1.00	0.1%	0.2%	0.1%	\$28,457,043,706	\$68,071,779,600	
★ 4	USD Coin USDC	\$1.00	-0.0%	0.1%	0.0%	\$3,617,295,314	\$47,395,206,765	
★ 5	Binance BNB	\$286.63	0.3%	1.5%	4.6%	\$449,782,486	\$46,816,396,118	
★ 6	XRP XRP	\$0.450858	1.2%	-2.1%	-8.7%	\$1,543,886,107	\$22,526,702,644	
★ 7	Binance USD BUSD	\$0.998936	-0.1%	-0.1%	-0.3%	\$5,208,296,894	\$20,892,269,192	
★ 8	Cardano ADA	\$0.424023	0.4%	-0.1%	-5.2%	\$372,393,621	\$14,352,117,337	
★ 9	Solana SOL	\$32.78	0.7%	0.9%	1.4%	\$525,253,926	\$11,650,588,428	

Why should I care about Blockchain?

Coins: 11,289 Exchanges: 940 Market Cap: \$1.605T ▲ 3.3% 24h Vol: \$124.858B Dominance: BTC 51.0% ETH 16.7% Gas: 69 GWEI

[Login](#) [Sign Up](#)



CoinGecko

Cryptocurrencies

Exchanges

NFT

Learn

Products



Candy

Portfolio

Search

/

Cryptocurrency Prices by Market Cap

The global cryptocurrency market cap today is \$1.6 Trillion, a ▲ 3.3% change in the last 24 hours. [Read more](#)

Highlights

Cryptocurrencies Highlights Chains Categories BCR-20 NFT Marketplace Layer 1 (L1) Customise

▲ #	Coin	Price	1h	24h	7d	24h Volume	Market Cap	Last 7 Days
-----	------	-------	----	-----	----	------------	------------	-------------

☆ 1	Bitcoin BTC	Buy \$41,819.61	▲ 0.4%	▲ 5.8%	▲ 13.2%	\$39,895,639,160	\$818,298,626,525	
-----	-------------	---------------------------------	--------	--------	---------	------------------	-------------------	--

☆ 2	Ethereum ETH	Buy \$2,223.67	▲ 0.2%	▲ 2.9%	▲ 10.8%	\$30,225,622,585	\$267,741,606,673	
-----	--------------	--------------------------------	--------	--------	---------	------------------	-------------------	--

☆ 3	Tether USDT	\$1.00	▼ 0.1%	▼ 0.0%	▲ 0.1%	\$68,136,345,602	\$89,759,699,960	
-----	-------------	--------	--------	--------	--------	------------------	------------------	--

☆ 4	BNB BNB	Buy \$230.62	▲ 0.2%	▲ 1.6%	▲ 2.3%	\$2,324,803,719	\$35,597,510,015	
-----	---------	------------------------------	--------	--------	--------	-----------------	------------------	--

☆ 5	XRP XRP	Buy \$0.618306	▼ 0.1%	▼ 0.0%	▲ 3.1%	\$1,701,445,098	\$33,379,898,878	
-----	---------	--------------------------------	--------	--------	--------	-----------------	------------------	--

☆ 6	Solana SOL	Buy \$60.48	▼ 0.3%	▼ 3.2%	▲ 11.4%	\$2,190,870,171	\$25,663,673,002	
-----	------------	-----------------------------	--------	--------	---------	-----------------	------------------	--

☆ 7	USDC USDC	\$1.00	▲ 0.0%	▲ 0.0%	▲ 0.2%	\$16,611,399,683	\$24,387,267,584	
-----	-----------	--------	--------	--------	--------	------------------	------------------	--

arch

RESEARCH



imgflip.com

Scientific Curiosity

Why should I care about Blockchain?

Understanding the Risks and Threats



THE VALUE OF
LEVERAGING
FULL-SPECTRUM
CYBER TO NEUTRALIZE
ENEMY THREATS.

THE VALUE OF PERFORMANCE.
NORTHROP GRUMMAN

MORE



Understanding Investments
(for Tendies ;-)



Job Opportunities

The future of Money?

A digital euro

The digital euro would be like euro banknotes, but digital. It would be an electronic form of money, issued by the Eurosystem (the ECB and the national central banks of the euro area), and would be accessible to all citizens and firms.

A digital euro would not replace cash, but rather complement it. A digital euro would give people an additional choice about how to pay and make it easier to do so, contributing to accessibility and inclusion.

FAQs on digital euro



Our work aims to ensure that in the digital age citizens and firms continue to have access to the safest form of money, central bank money.

Christine Lagarde, President of the ECB



SBA
Research

Is there Science in Blockchain?



universität
wien

SBA
Research

Is there Science in Blockchain?

Google Scholar

blockchain

Articles About 555.000 results (0,03 sec)

Any time [PDF] Blockchain challenges and opportunities: A survey [PDF] allquantor.at
Z Zheng, S Xie, HN Dai, X Chen... - International journal of web ..., 2018 - allquantor.at
... survey on the **blockchain** technology. In particular, this paper gives the **blockchain** taxonomy,
introduces typical **blockchain** consensus algorithms, reviews **blockchain** applications and ...
☆ Save 99 Cite Cited by 2717 Related articles All 14 versions Web of Science: 976

Sort by relevance [PDF] ieee.org
Sort by date
Any type [PDF] ieee.org
Review articles
 include patents
 include citations
 Create alert

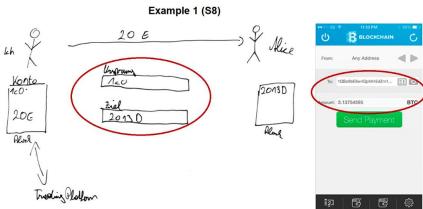
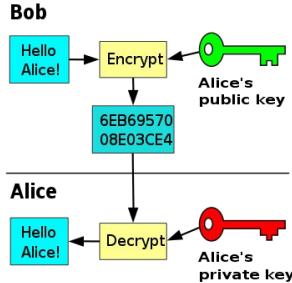
A survey of **blockchain** from the perspectives of applications, challenges, and opportunities [PDF] ieee.org
AA Monrat, O Schelén, K Andersson - IEEE Access, 2019 - ieexplore.ieee.org
... of the tradeoffs of **blockchain** and also explains the taxonomy and architecture of **blockchain**,
provides a ... In addition, this paper also notes the future scope of **blockchain** technology. ...
☆ Save 99 Cite Cited by 352 Related articles All 6 versions Web of Science: 140

A vademecum on **blockchain** technologies: When, which, and how [PDF] sorbonne-univ
M Belotti, N Božić, G Pujolle... - ... Surveys & Tutorials, 2019 - ieexplore.ieee.org
... to better provide details fundamental for the possible **blockchain** platform choice. Section V
starts our **blockchain** vademecum, about When to use **blockchain**, Which solution to use and ...
☆ Save 99 Cite Cited by 270 Related articles All 9 versions Web of Science: 102



universität
wien

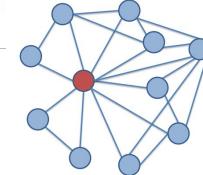
SBA
Research



Cryptography

Usability

Security



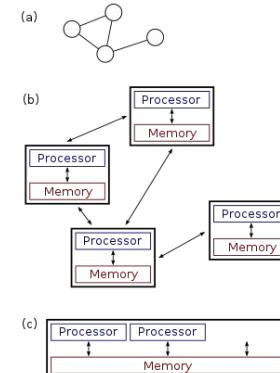
Is there Science in Blockchain?



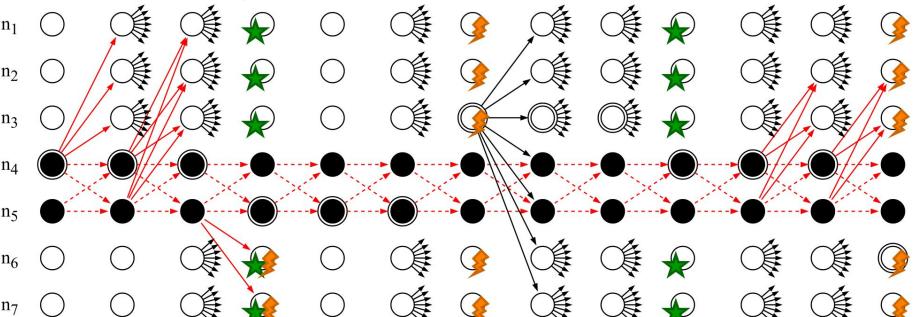
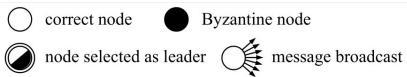
Privacy

Economics & Game Theory

p.d	SELLER	
	COOPERATE	DEFECT
BUYER	COOPERATE	
	DEFECT	



Distributed Computing



DLT research is exciting!





DLT Research needs YOU!



universität
wien

SBA
Research

Introduction to Cryptocurrencies & DLT

- Terminology can be ambiguous
- Depending on the discipline (e.g., computer science, accounting & legal studies, economics, etc.) the terminology may be defined/interpreted differently
- Terminology is experiencing changes in its utilization and understanding due to fast paced (technological) progress



Characteristics of this Research Field

Field =
Cryptocurrency and
Distributed Ledger Technologies

- Community driven
- An Interdisciplinary topic
- (still) A hype topic ?



Characteristics of this Research Field

Community Driven:

- Quick online publication and discussion
- Rapid free software development
- Strong distributed systems without trusted 3rd party (TTP) mentality and mindset
- Also a lot of start-ups, companies but also scams
- ...



Characteristics of this Research Field

Interdisciplinary:

- Cryptocurrencies attract many different communities [1]
 - technically interested, ideologists, start-ups, large enterprises, public authorities, banks, financial regulators, business men, investors, criminals, ...
- Cryptocurrencies have many aspects
 - economical, legal, sociological, game theoretic, usability, technological, ...
 - Technological: network & distributed systems, cryptography, language security, ...

[1] [yelowitz2015characteristics, krombholz2016other]

Characteristics of this Research Field

A Hype Topic:



There are currently about ~~~700~~ cryptocurrencies

https://blog.adafruit.com/wp-content/uploads/2014/03/adafruit_2738.jpg

<https://www.coingecko.com/en>

The hype

Mainframe-Migration:
GFS für Linux

DOST
DEUTSCHE
OPENSTACK TAGE

21. & 22. JUNI KÖLN

Plätze sichern u.
openstack-tage.de/a

MAGAZIN FÜR PROFESSIONELE
INFORMATIONSTECHNIK

6 JUNI 2016

Ein bisschen mehr Stand:
Cloud-Marktplatz

Authentifizierung, Bitcoin, Industrie 4.0, DRM, ...

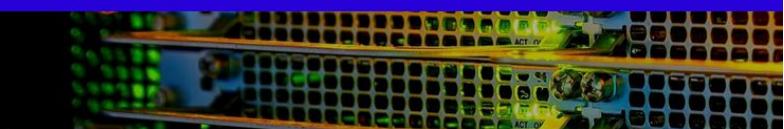
WIR ED

SUBSCRIBE

PARTNER CONTENT KARIAPPA BHEEMIAH, GRENOBLE ECOLE DE MANAGEMENT.

BLOCK CHAIN 2.0: THE RENAISSANCE OF MONEY

Bloomberg Businessweek Markets Tech Pursuits Politics Opinion Businessweek



Blockchain Goes Beyond Crypto-Currency

Businesses are exploiting a technology created for bitcoin.

THE WALL STREET JOURNAL.

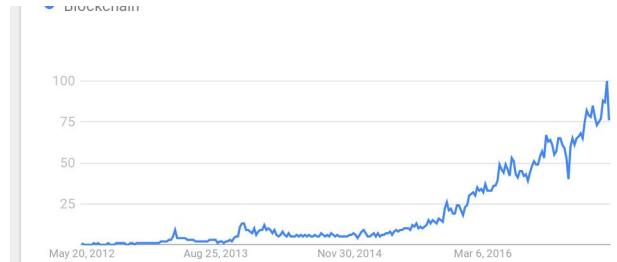
Subscribe | Sign In

JOURNAL REPORTS: LEADERSHIP

What Blockchain Is and What It Can Do

Digital Asset CEO Blythe Masters on how the technology works in transactions

Tech Trends



Forbes

Tech / #BigData

MAY 27, 2016 @ 02:46 AM 25,923 VIEWS

How Blockchain Technology Could Change The World

UNIVERSITÄT
WIEN

VUB
Research

The hype

A collage of news snippets and images related to blockchain technology, illustrating its current status as a buzzword or 'hype'.

Left Column (Blockchain News Snippets):

- GSX Logo:** GSX (Gesellschaft für Systeme und X) logo, featuring a large stylized 'X' and 'S'.
- Deutsche OpenStack Tage (DOST):** Logo for the Deutsche OpenStack Tage conference, held on June 21-22 in Cologne.
- Wall Street Journal Article:** Headline: "What Blockchain Is and What It Can Do".
- the guardian Article:** Headline: "Blockchain: the answer to life, the universe and everything?"
- Bloomberg Businessweek Article:** Headline: "Blockchain Goes Beyond Crypto-Currency". Subtext: "Businesses are exploiting a technology created for bitcoin."

Bottom Right (University Logos):

- University of Vienna Research:** Logo for the University of Vienna Research.

Bottom Left (Text):

Businesses are exploiting a technology created for bitcoin.

The hype

This was in 2016
Has anything changed?

A screenshot of a news article from Bloomberg Businessweek. The headline reads "Blockchain: the answer to life, the universe and everything?". The sub-headline says "Technology Half full: solutions, innovations, answers". The article is categorized under "Technology" and "BigData". It includes a photo of a server rack.

Blockchain Goes Beyond
Crypto-Currency

Businesses are exploiting a technology created for bitcoin.

Tech / #BigData

MAY 27, 2016 @ 02:46 AM 25,923 VIEWS

How Blockchain Technology Could
Change The World



CRYPTOCURRENCY

Kim Kardashian to pay \$1.26 million SEC crypto case

The Securities and Exchange Commission has filed a civil complaint against Kim Kardashian West, her husband, and their company for their conduct on Instagram.

FINTECH:

Beware the hype over central bank digital currencies



TECH FIX

What's All the Hype About the Metaverse?

Microsoft cited the metaverse as a reason for buying Activision Blizzard for \$68.7 billion. Let's break down what that really means.

I guess not ;)

ENTERPRISE TECH

NFTs: What The Hype Is About And Where They Are Headed

Boards, Policy & Regulation | Technology | ESG Investors | Boards | Corporate Governance

Sam Bankman-Fried convicted of multi-billion dollar FTX fraud

By Luc Cohen and Jody Godoy
November 3, 2023 9:43 AM GMT+1 · Updated 25 days ago

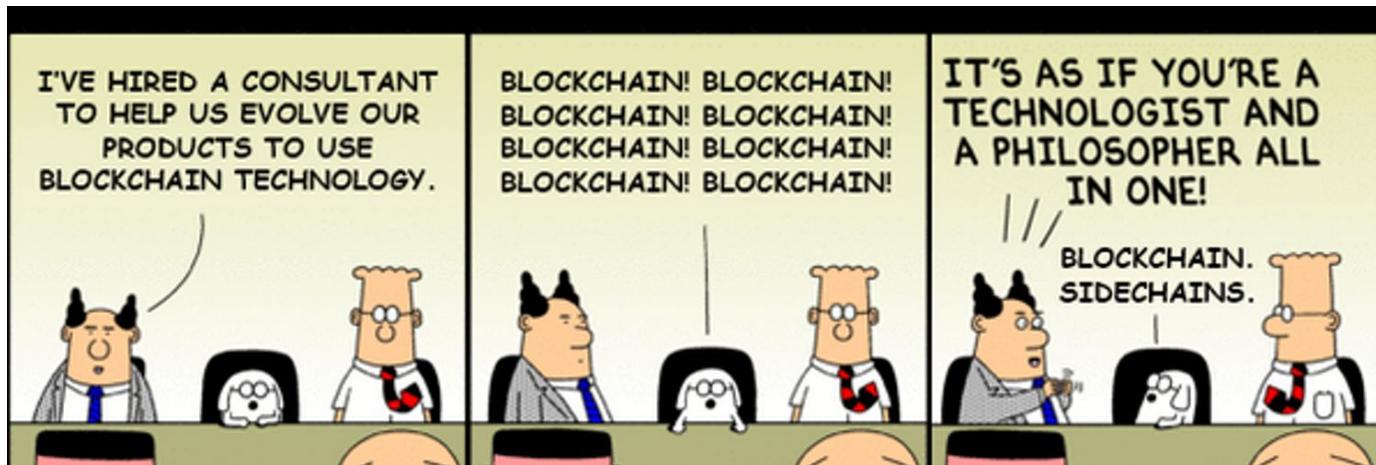


published author.

Become an FT subscriber to read:
Traders lend out cryptocurrencies in quest for huge returns

Characteristics of this Research Field

A Hype Topic ?:



Mashup from Scott Adams cartoon by unknown author



universität
wien

SBA
Research

Origins of the Term Cryptocurrency



Satoshi actually does not mention Cryptocurrency!

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending.



Electronic Cash in the Bitcoin Whitepaper

It is highly unlikely that the use of the term "electronic cash" in the Bitcoin whitepaper was inspired by the following:

"Electronic cash was, until 2007, the debit card system of the German Banking Industry Committee, the association that represents the top German financial interest groups." [1]

It is much more likely that Nakamoto was conceptually linking the idea to works such as:

D. Chaum and S. Brands, "'Minting' electronic cash," in IEEE Spectrum, vol. 34, no. 2, pp. 30-34, Feb. 1997, doi: 10.1109/6.570825.



Image Source: <https://www.ec-cash-direkt.de/kartenzahlung/ec-cash.html>

[1] https://en.wikipedia.org/wiki/Electronic_cash

Earliest cited reference we could find that uses the term Cryptographic Currency (System):

- 2007 in “León, Xavier, and Leandro Navarro. *Currency management system: A distributed banking service for the grid.* Technical Report “

Early use of the term Cryptocurrency in the context of Bitcoin/Blockchain:

- 2010 by Satoshi Nakamoto in the release message for version 0.3 of the Bitcoin software "Announcing version 0.3 of Bitcoin, the P2P cryptocurrency!..." "

see: <https://satoshi.nakamotoinstitute.org/emails/bitcoin-list/threads/12/>

Technical Terms in the context of Cryptocurrencies can be imprecise!

- . In Computer Science there are a variety of terms being used within technical publications, e.g., "**electronic coin**", "**electronic cash**", "**e-cash**", "**electronic money**", "**e-money**", "**electronic currency**", "**digital currency**", "**digital cash**", "**virtual currency**" and of course "**cryptocurrency**" to refer to systems handling some form of transaction processing for digital payments.
- . Hereby, often no explicit distinction is made between these terms and they are used synonymous for the purpose of illustrating the intended technical functionality – even if they have different legal or economic connotations.

The Term Cryptocurrency

- No universally agreed upon definition
- Some common properties
 - “is a **medium of exchange** using cryptography to secure the transactions and to control the **creation of additional units**” [1]
 - “**decentralized** control” [1,2]
 - “Bitcoin is an open-source, peer-to-peer **digital currency** ... it is the world’s first completely **decentralized** digital-payments system” [3]

[1] https://en.wikipedia.org/wiki/Cryptocurrency#cite_note-crypto_currency-1

[2] http://www.trssllc.com/wp-content/uploads/2013/05/White_Paper_Bitcoin_101.pdf

[3] https://www.mercatus.org/system/files/Brito_BitcoinPrimer.pdf

The Term Cryptocurrency

- Many sources now define cryptocurrency in the context of decentralized cryptographic currencies – tying the term closer to Bitcoin and Bitcoin-like protocols.
- *Cryptocurrency*:
“Cryptocurrencies too must have security measures that **prevent** people from **tampering with the state of the system**, and from equivocating, that is, making mutually inconsistent statements to different people”

[narayanan2016bitcoin]



Standardization Definitions of Cryptocurrency

ISO 22739:2020 Definitions:

Cryptocurrency “crypto-asset designed to work as a medium of value exchange

- Note 1 to entry: Cryptocurrency involves the use of decentralized control and cryptography to secure transactions, control the creation of additional assets, and verify the transfer of assets.”

Crypto-asset “digital asset implemented using cryptographic techniques”

Digital asset “asset that exists only in digital form or which is the digital representation of another asset”

Asset “anything that has value to a stakeholder”



Cryptocurrency = Kryptowährung?

- Translations of specific (technical) terms in different languages, e.g. English to German and vice versa, may not always have exactly the same meaning!
- Some terms are generally not translated (e.g. staking, blockchain, hard fork, airdrops) while others are
- The meaning of terms can actually change over time!
- In practice, the term cryptocurrency sometimes only refers to the money-like properties of the assets, however other times the entire technical system is addressed



Cryptocurrency = Kryptowährung?

From Ökosoziales Steuerreformgesetz 2022 Teil I (ÖkoStRefG)

§ 27b. EStG

(4) Eine Kryptowährung ist eine digitale Darstellung eines Werts, die von keiner Zentralbank oder öffentlichen Stelle emittiert wurde oder garantiert wird und nicht zwangsläufig an eine gesetzlich festgelegte Währung angebunden ist und die nicht den gesetzlichen Status einer Währung oder von Geld besitzt, aber von natürlichen oder juristischen Personen als Tauschmittel akzeptiert wird und die auf elektronischem Wege übertragen, gespeichert und gehandelt werden kann (§ 2 Z 21 des Finanzmarkt-Geldwäschegegesetzes).“

- We commented on the tax law changes:

https://www.parlament.gv.at/PAKT/VHG/XXVII/SNME/SNME_110826/index.shtml

see https://www.ris.bka.gv.at/Dokumente/BgbIAuth/BGBLA_2022_I_10/BGBLA_2022_I_10.html

Cryptocurrency = Kryptowährung?

ÖkoStRefG definition of "Kryptowährung" is now the same as the definition of virtual currencies (Virtuelle Währungen) from the the 5th anti-money laundering Directive of the EU, which can also be found in other Austrian laws, e.g.,

Finanzmarkt-Geldwäschegegesetz (FM-GwG),

Bilanzbuchhaltungsgesetz 2014 (BiBuG 2014)

Wirtschaftstreuhandberufsgesetz 2017 (WTBG 2017)

→ Does this mean virtual currencies = Kryptowährung?



Cryptocurrency = Kryptowährung?

- What about Digital Assets that do not meet the definition of virtual currencies?
- Are Non-fungible Tokens such as CryptoKitties included?
- What about Tokens used for testing purposes (like the ones you will receive in the challenge environment)

New Regulations from the EU such as Markets in Crypto Assets (MiCA) now more generically refer to **Crypto-assets (Kryptowerte bzw. Krypto-Assets) instead of virtual currencies.**

- This terminology makes the focus on the monetary-like aspects of the assets a lot clearer than the term cryptocurrency

*A discussion on different uses of such terminology in Austrian legislature can be found as part of the following diploma thesis:

E. Putz, Der behördliche Zugriff auf Krypto-Assets (2023),
<https://epub.jku.at/obvulihs/content/titleinfo/8486657>



Blockchain Babel?



Image: Großer Turmbau zu Babel von Pieter Bruegel, 1563, Kunsthistorisches Museum

The term Cryptocurrency and Crypto-asset

In this presentation we will use cryptocurrency and crypto-assets as generalized technical terms

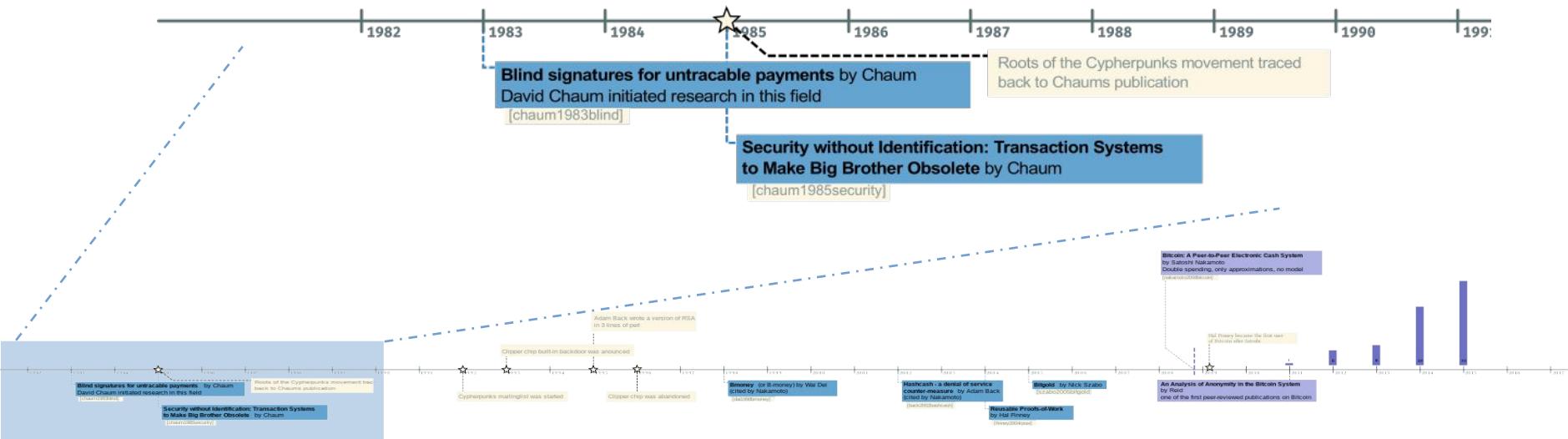
- **crypto-asset** intends to describe some form of digital asset, where ownership and transfers thereof are recorded on distributed ledgers and these processes are secured by cryptographic primitives
- **cryptocurrency** intends to describe the entire system that realizes the functionality of allowing crypto-assets to be secured and transferred, including its technical components



A Brief History of Cryptocurrencies



Early Origins of Cryptocurrencies



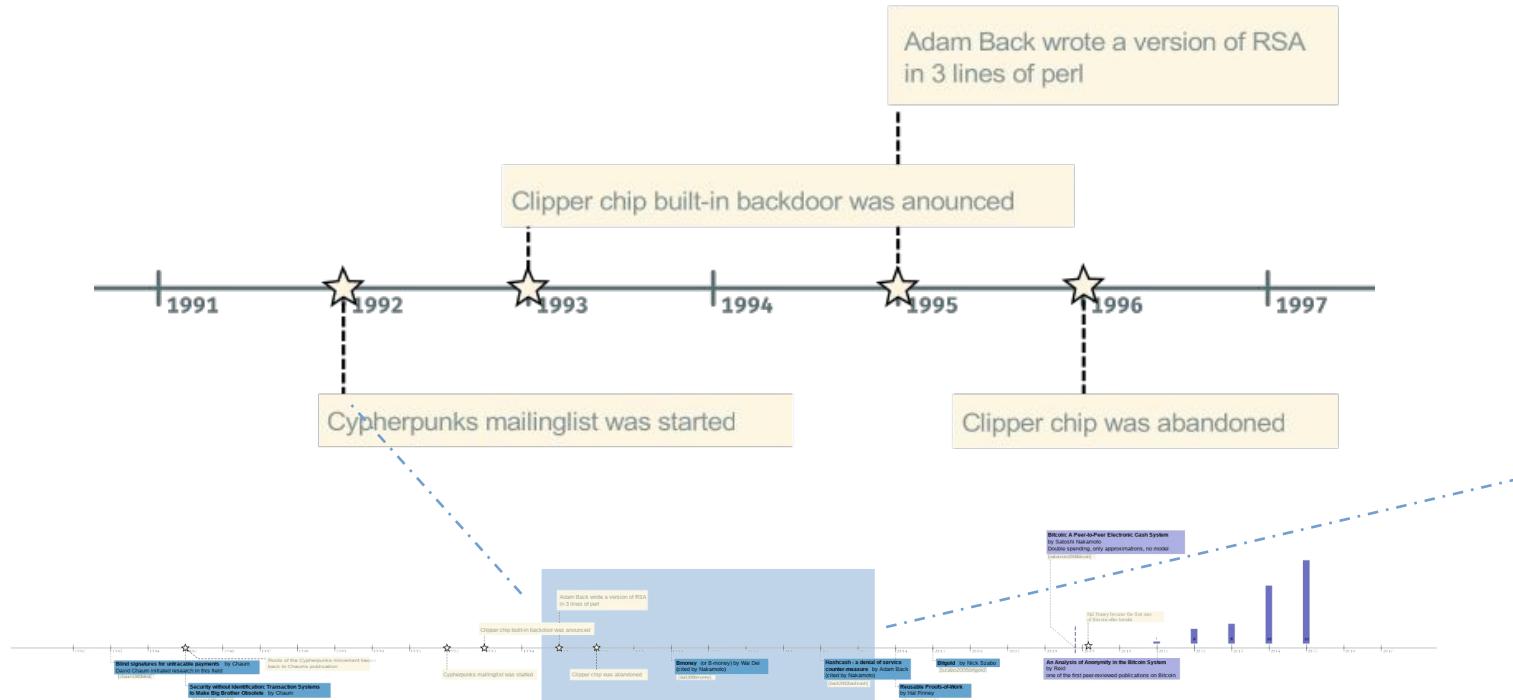
Early Origins of Cryptocurrencies

“first wave” of cryptocurrencies pioneered the idea to rely on cryptography for building systems that aim to achieve cash-like properties in digital money

- Pioneered by David Chaum
 - . [chaum1983blind]
 - Chaum, D. “Blind signatures for untraceable payments”, 1983
- Relies on blind signatures
- Bank-issued cash – Needs Trusted Third Party
- Centralized issuance
- Important design goal was **privacy**
- Origins of the cypherpunks movement



Crypto Wars



Crypto Wars

RSA in the shortest amount of perl code

(Challenge started by Adam Back)



```
#!/bin/perl -sp0777i<X+d*1MLa^*1N%0]dsXx++1MlN/dsM0<j]dsj
$/=unpack('H*',$_);$_=`echo 16dio\U$k"SK$/SM$n\Esn0p[1N*1
1K[d2%Sa2/d0$^IxP"|\dc`;s/\W//g;$_=pack('H*',/((..)*$)/)
```

Crypto Wars

RSA in the shortest amount of perl code

(Challenge started by Adam Back)

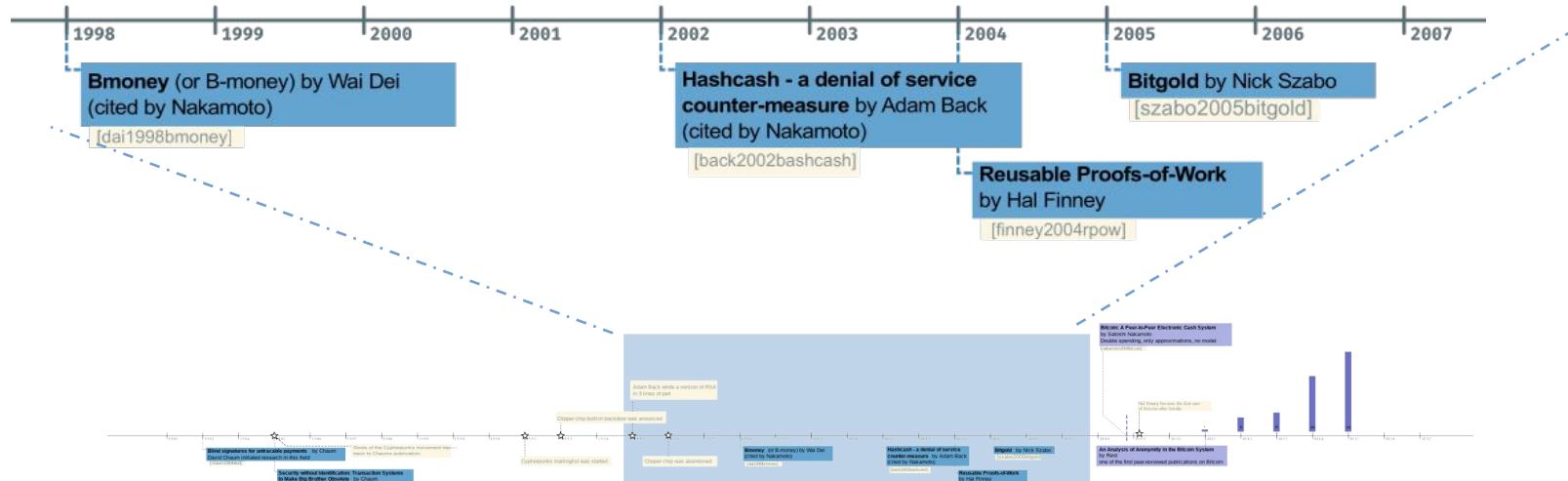
PRESS RELEASES

U.S. Treasury Sanctions Notorious Virtual Currency Mixer
Tornado Cash

```
# !/usr/bin/perl
$/= pack("C", 0xa2/d0$^Ixp" | dc` ; s/\W//g; $_=pack('H*', /((..)*)$/))
```



“Second Wave” Cryptographic Currencies



“Second Wave” Cryptocurrencies

The idea that “**proof-of-work**” can be used to add scarcity/cost to digital tokens and serve as a basis for cryptocurrencies is beginning to take shape

- Also addresses question of how to avoid centralized issuance

Implementations still relied either on **central elements** (TTP) to maintain ownership records or unrealistic system assumptions

- . Bmoney (or B-money) by Wai Dai [dai1998bmoney]
- . Hashcash by Adam Back [back2002hashcash]
- . Reusable Proofs-of-Work by Hal Finney [finney2004rpow]
- . Bit gold by Nick Szabo [szabo2005bitgold]



Bitcoin and Beyond

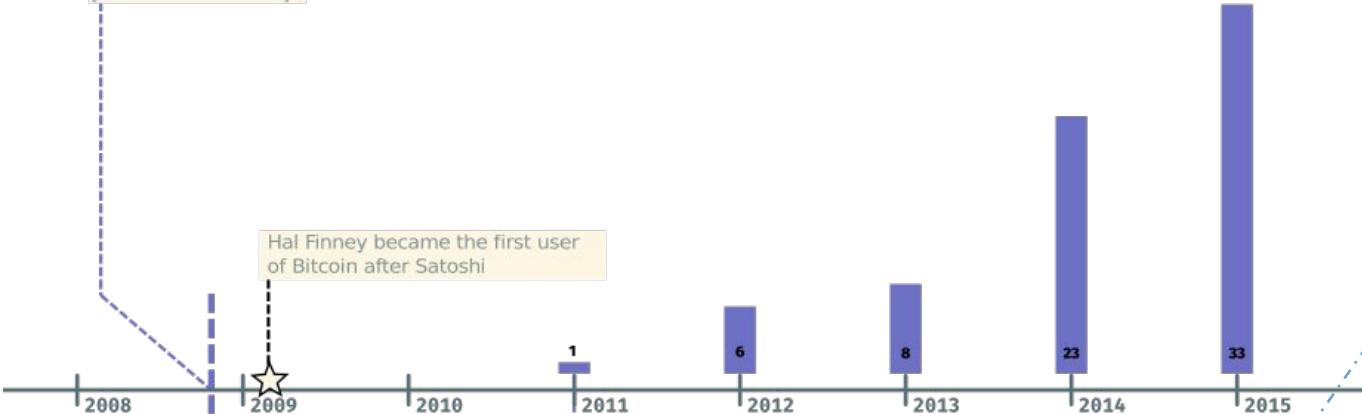
Bitcoin: A Peer-to-Peer Electronic Cash System

by Satoshi Nakamoto

Double spending, only approximations, no model

[nakamoto2008bitcoin]

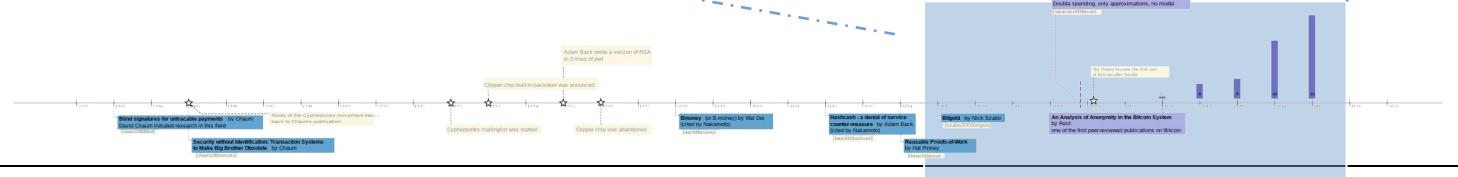
Hal Finney became the first user of Bitcoin after Satoshi



An Analysis of Anonymity in the Bitcoin System

by Reid and Harrigan

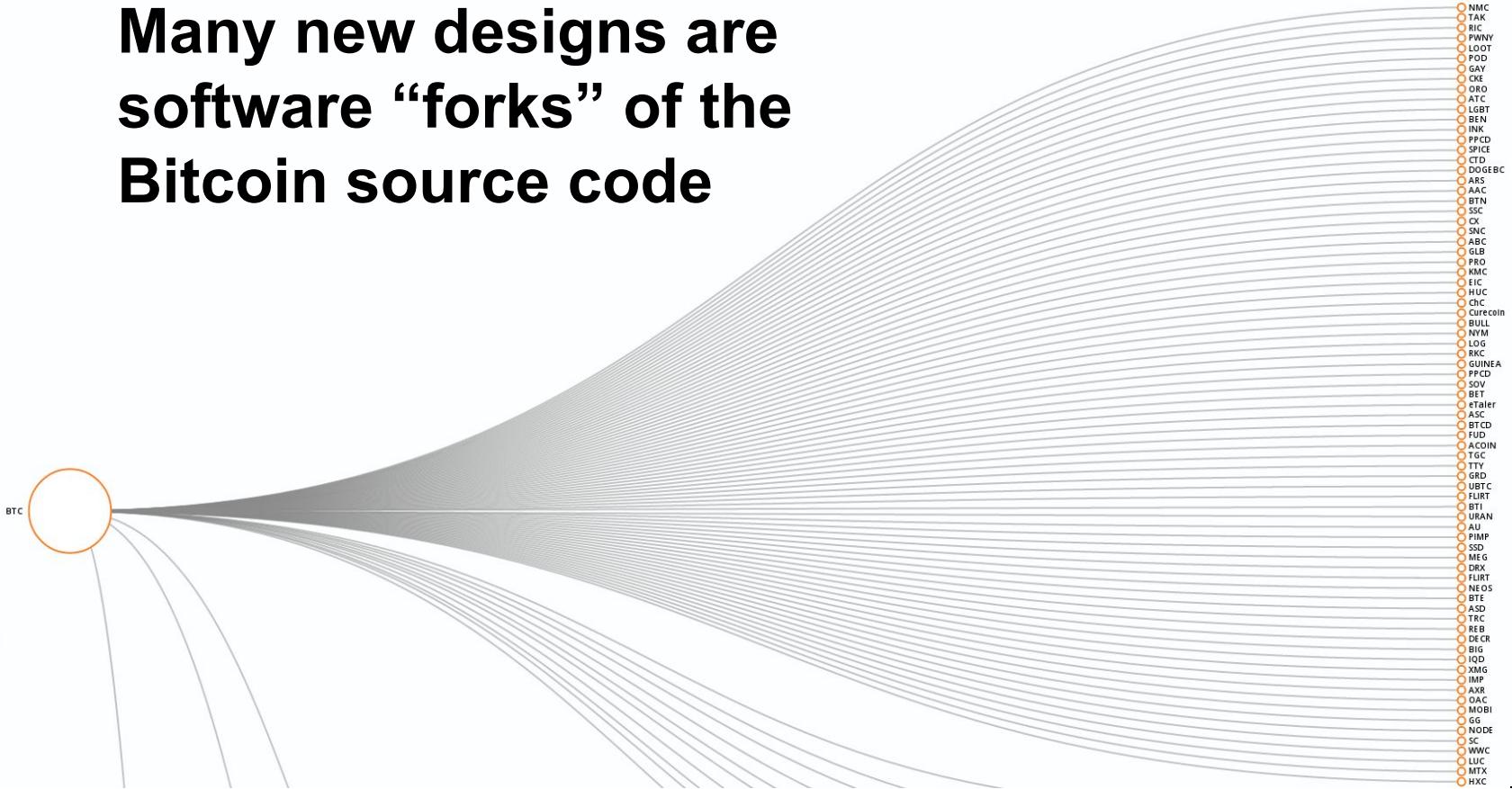
one of the first peer-reviewed publications on Bitcoin



universität
wien

SBA
Research

Many new designs are software “forks” of the Bitcoin source code



source: <http://mapofcoins.com/bitcoin#>

Problems and Challenges for Cryptocurrency Designs



Challenges when designing a Cryptocurrency:

- Issuer is often centralized
- Privacy and Traceability
- Double Spending
- Decentralization
- Scalability
- Legal and Economic Issues



Problems and Challenges for Cryptocurrencies

Centralized Issuance

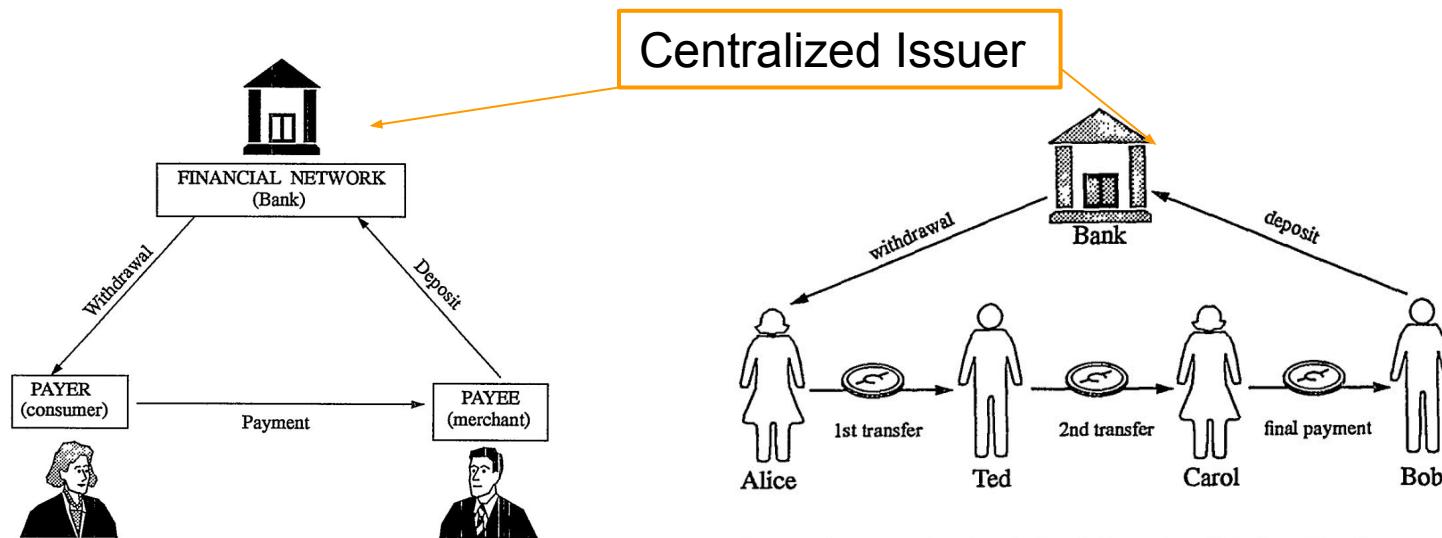


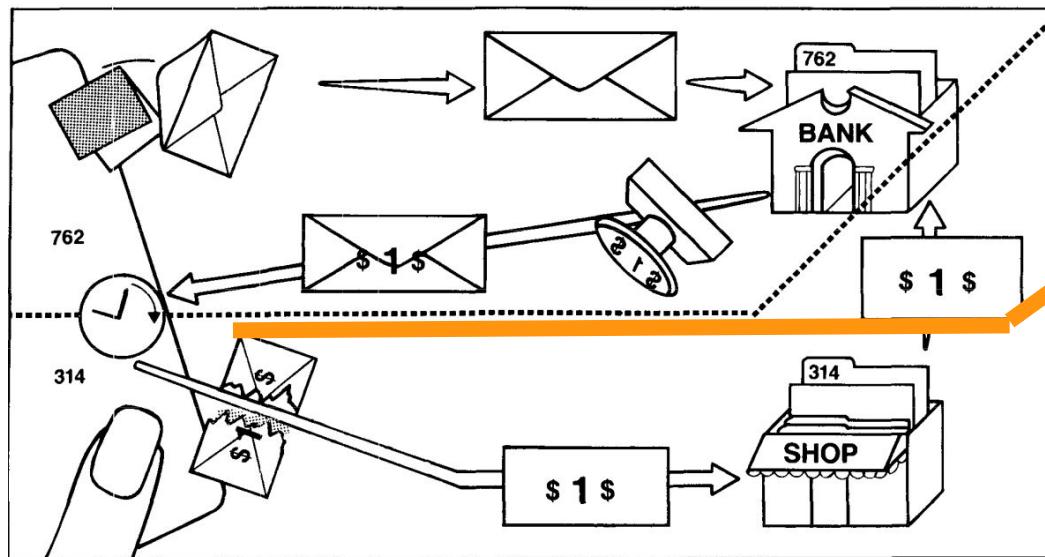
Figure 2. A maximum length path of a coin in a system which allows 2 transfers per coin.

Figure 1. The three types of transactions in a basic electronic cash model.

Law, Laurie, Susan Sabet, and Jerry Solinas. "How to make a mint: the cryptography of anonymous electronic cash." Am. UL Rev. 46 (1996): 1131. (note: Research from NSA employees)

Problems and Challenges for Cryptocurrencies

Preventing Traceability



Prevent linking
user withdrawals
and spending

Chaum, David. "Security without identification: Transaction systems to make big brother obsolete." Communications of the ACM 28, no. 10 (1985): 1030-1044.

Problems and Challenges for Cryptocurrencies

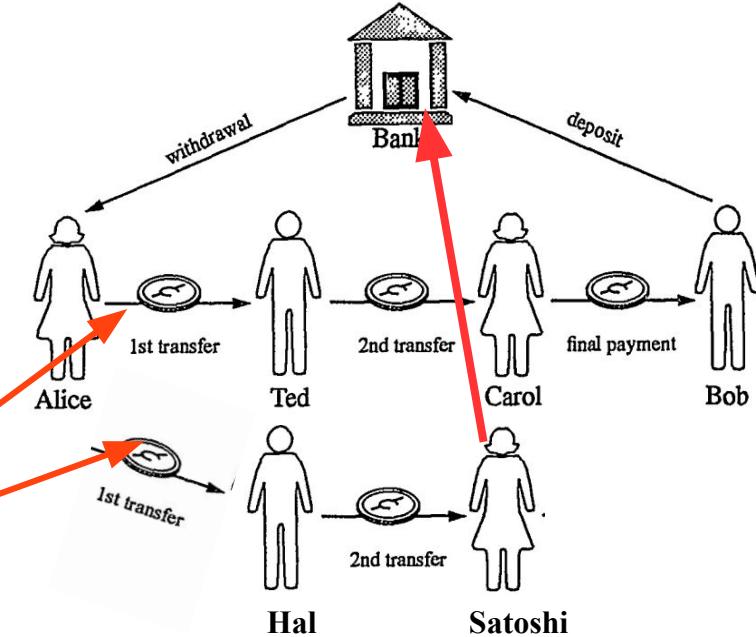
Double Spending

How can we prevent Alice from spending her coin to multiple parties?

Simple Solution:

- Ask the Bank if coin was spent
 - Requires being online
 - Relies on trusted third party
 - Problematic for privacy

Double
spend



Law, Laurie, Susan Sabetta, and Jerry Solinas. "How to make a mint: the cryptography of anonymous electronic cash." Am. UL Rev. 46 (1996): 1131. (note: Research from NSA employees)

The Double-Spending Problem

The Double-Spending Problem

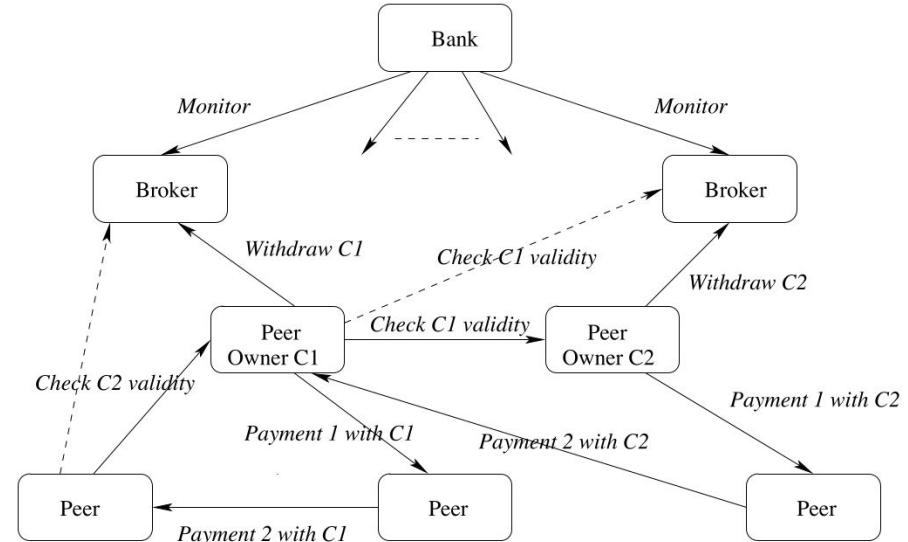
- Goal is to prevent anyone from spending the same (currency) units more than once
- We could write every transaction into a ledger and use the order of its entries to prevent double-spending
- If we have **consensus** on the ledger we can hence **solve** the **double-spending** problem
- Some participants may act maliciously (Byzantine failures)
- Solving distributed (Byzantine) consensus hence allows us to readily create a cryptocurrency by using it to agree on a ledger

Problems and Challenges for Cryptocurrencies

Decentralization (pre-Bitcoin)

Challenge: Prevent double-spending if there is no single Bank (TTP) to ask

- Requires some form of agreement (consensus) on which coins are spent
- Anonymity (e.g. of Brokers) makes this problem very hard

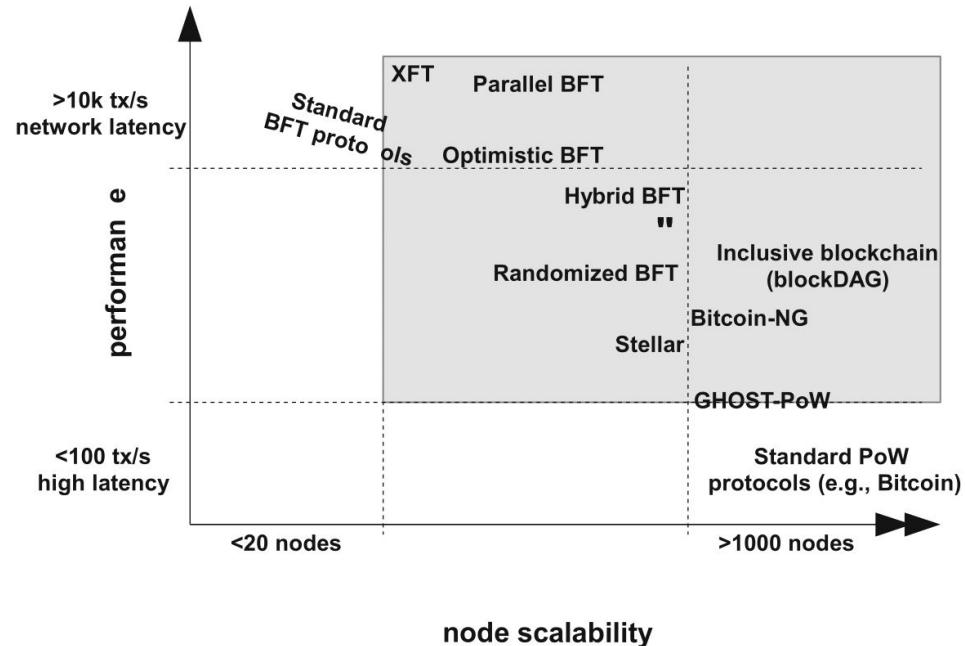


Problems and Challenges for Cryptocurrencies

Scalability

Challenge: More centralized designs with few consensus participants scale better than large-scale P2P systems, if they have to reach agreement

- "Classic" BFT protocols have message complexity of at least $O(n^2)$



Problems and Challenges for Cryptocurrencies

Legal and Economic Issues

- Early (centralized) digital currency systems often shut down due to legal concerns and alleged criminal activity
 - Many of these systems did not heavily rely on cryptography



Failed Economically



Shut Down due to legal disputes

<https://upload.wikimedia.org/wikipedia/commons/e/ec/DIGICASH.png>

<https://cs.stanford.edu/people/eroberts/cs201/projects/2010-11/Bitcoins/e-gold.html>

https://en.wikipedia.org/wiki/Liberty_Reserve#/media/File:Liberty_Reserve_logo.svg

<https://en.wikipedia.org/wiki/Beenz.com#/media/File:Beenz.com-logo.png>

Problems and Challenges for Cryptocurrencies

Legal and Economic Issues

- Legal classification continues to be a challenge



PoW & Bitcoin



- “Satoshi Nakamoto” (pseudonym)
 - S. Nakamoto, “*Bitcoin: A Peer-to-Peer Electronic Cash System*”, (2008) [nakamoto2008bitcoin]
 - + source code
- first user, created “genesis block”

Supply of Bitcoins

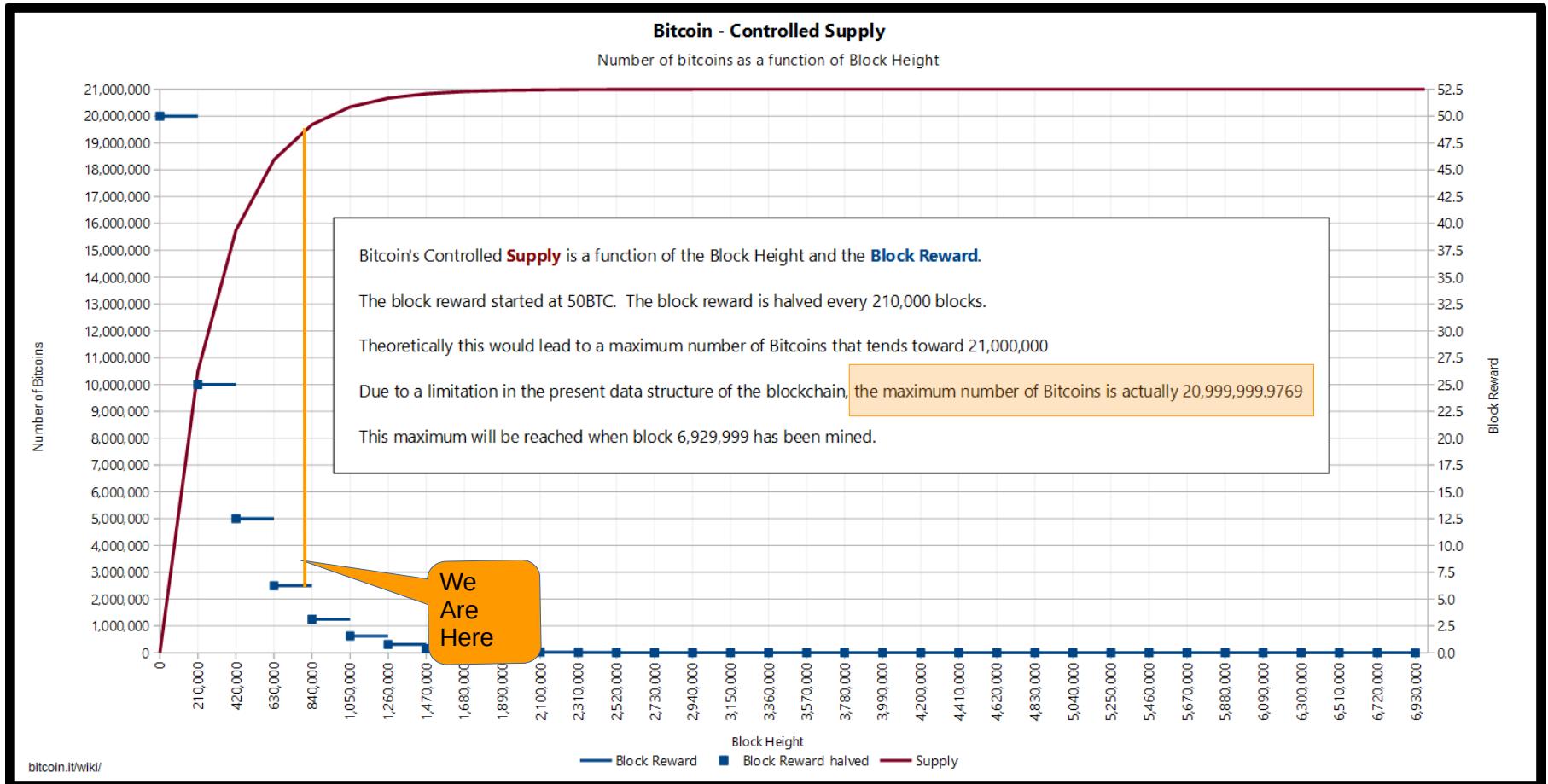
- 1 BTC = 10^8 Satoshi
- There is **no cryptographic** limit of 21 Million bitcoins
- It is a software limit agreed among users/miners of Bitcoin
- Rule: half the mining reward after all 210.000 blocks (~all 4 years)
 - Era 1: 50 50 BTC
 - Era 2: 50/2 = 25 BTC
 - Era 3: 25/2 = 12.5 BTC
 - Era 4: 12.5/2 = 6.25 BTC
 - ...
 - Era 33: **0.0000001 “Satoshis”** (smallest unit)
- Beginning with Era 34 no more block rewards are payed out – Fees Only
This will most likely happen around the first quarter of 2100, depending on hash rate increase

Supply of Bitcoins

Current maximum number of <i>bitcoins</i>	21,000,000 21×10^6 21 Million
Current maximum number of <i>satoshis</i>	2,100,000,000,000,000 2.1×10^{15} 2 Quadrillion (short scale) 2 Billiard (long scale)
Amount of USD in the world (M2) [1]	~ 10,000,000,000,000 ~ 10 Trillion (short scale) ~ 10 Billion (long scale)
Theoretical maximum of <i>satoshis</i> when fully utilizing the data type <i>signed int64_t</i>	9,223,372,036,854,775,807 $2^{64} / 2 - 1$ 9 Quintillion (short scale) 9 Trillion (long scale)

[1] <http://money.howstuffworks.com/how-much-money-is-in-the-world.htm>

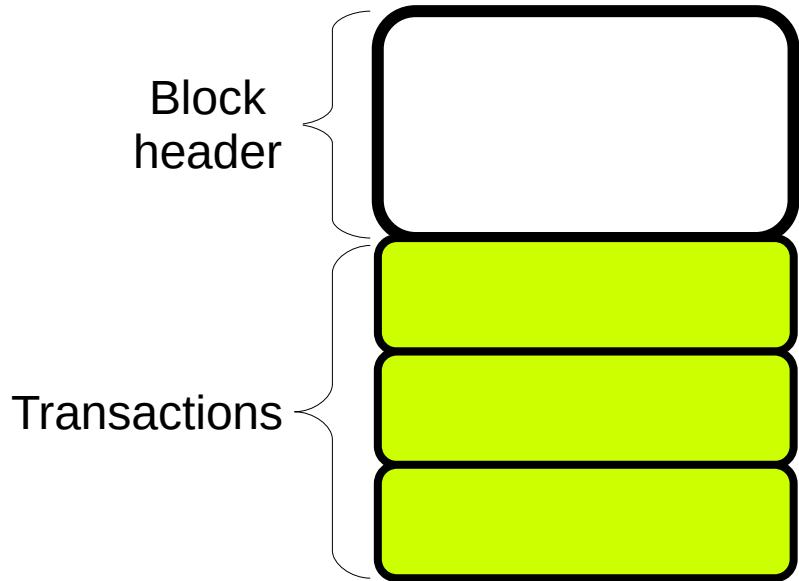
Supply of Bitcoins



Bitcoin Mining & Transactions

- Blocks
- Mining
- Addresses
- Transactions
- Bitcoin scripting language
- Transaction fees

Block



Transactions & Blocks

Transaction (tx)

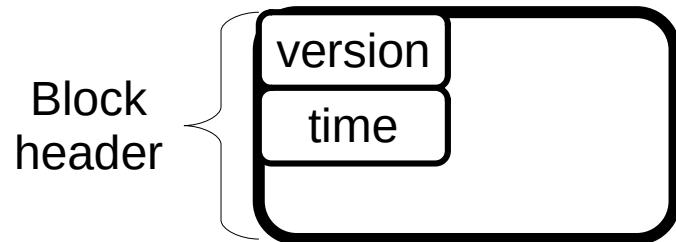
- can be created by *everyone who owns* currency units
- transfer currency units

Block (b)

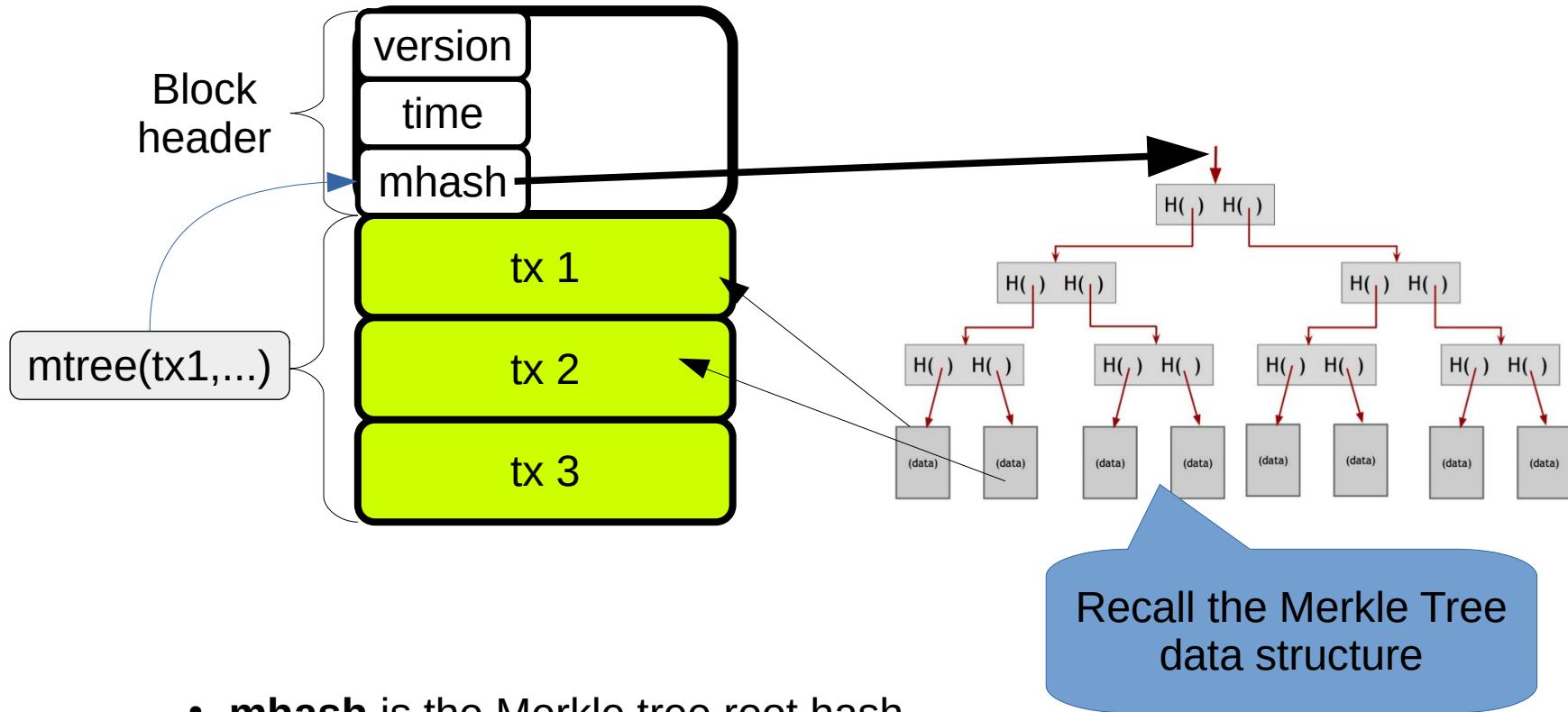
- encapsulates number of transactions
- created by *miners*, include **proof-of-work**
- chained together to **synchronized time line**

Both are disseminated to all Bitcoin users in the P2P network via a gossiping protocol

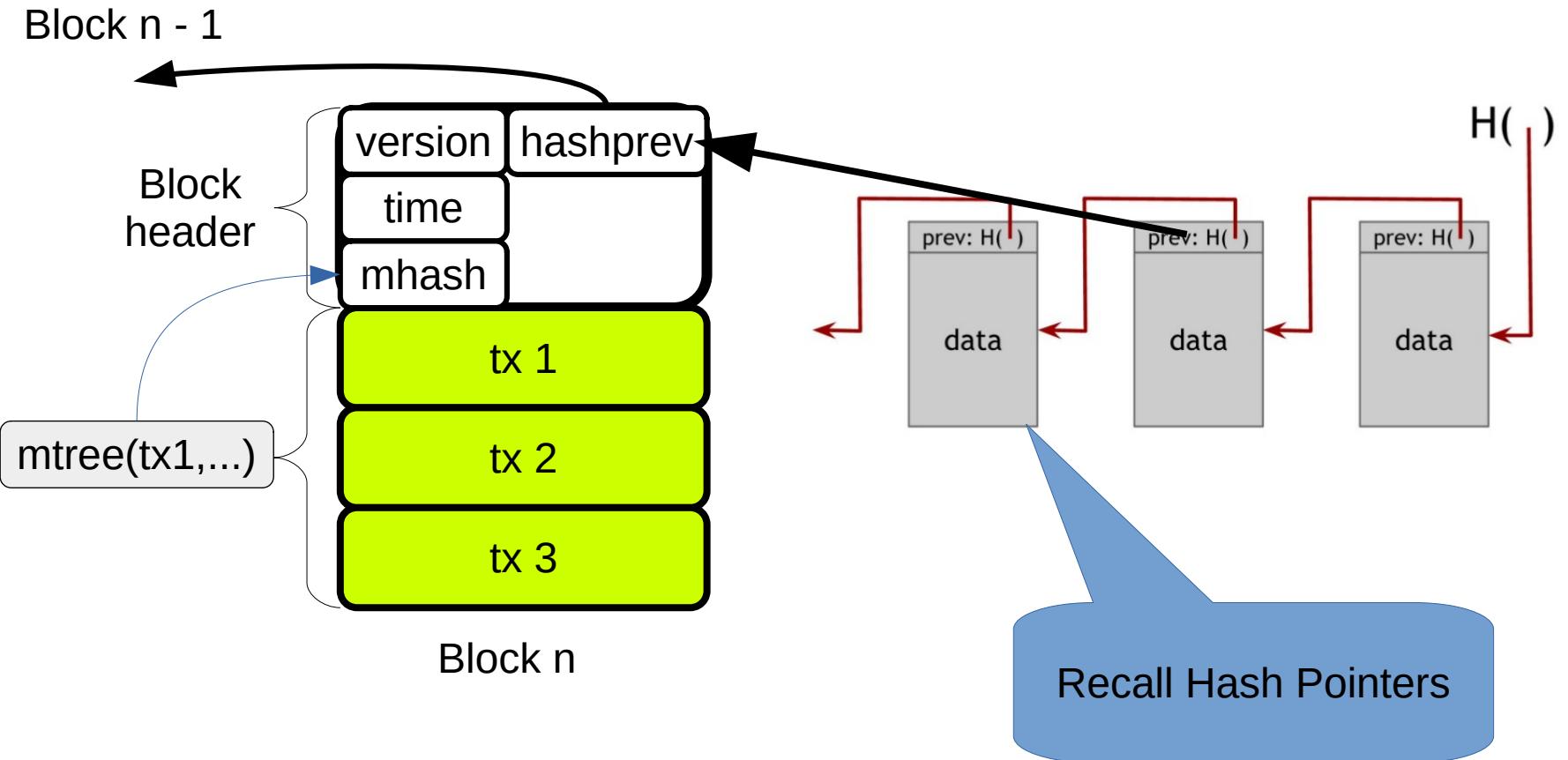
Block



Block



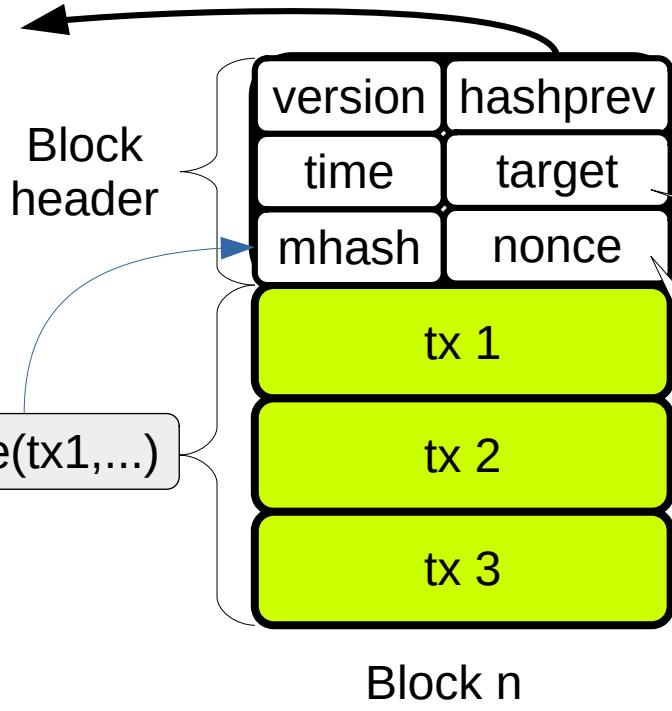
Block



- **mhash** is the Merkle tree root hash
- **mtree** is the Merkle tree of transactions that are included in the block
- **hashprev** is the double-sha256 hash of the previous block's header

Block

Block n - 1



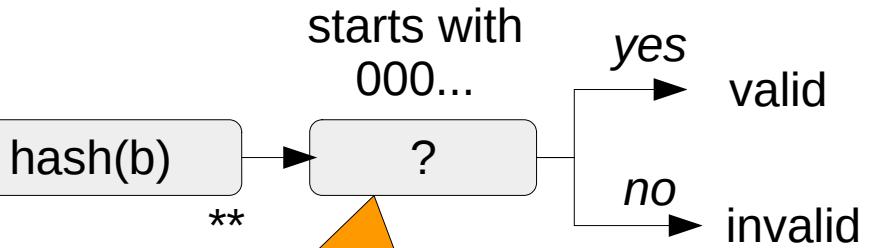
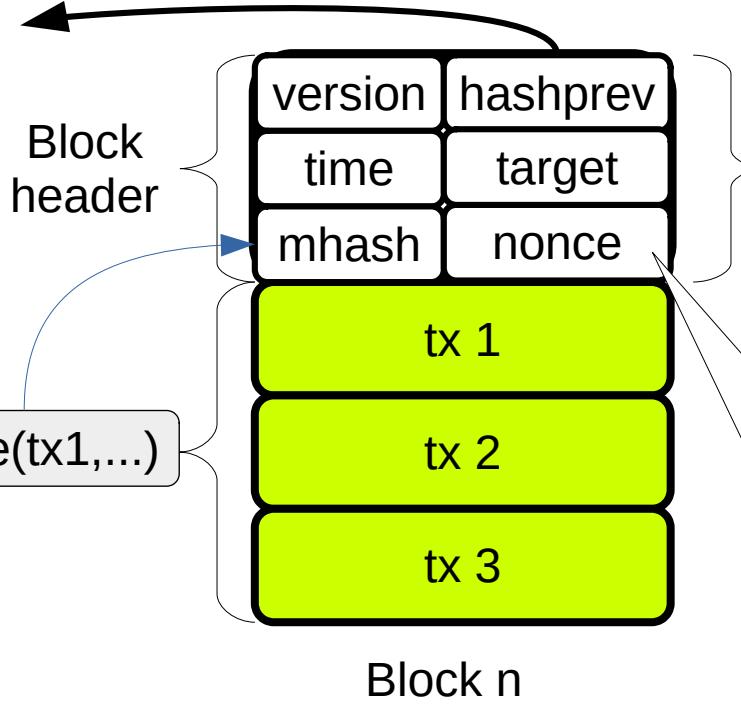
Defines the required number of zeros at the beginning of the PoW hash*

32 bit value can be
Changed to find PoW
via brute-force search

[*] Actually this is called *bits* or *nBits* and requires a conversion to get the target value

Mining

Block n - 1



Check if the hash of the Block header meets PoW requirements

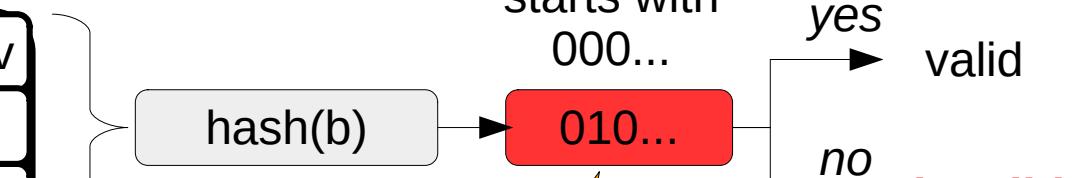
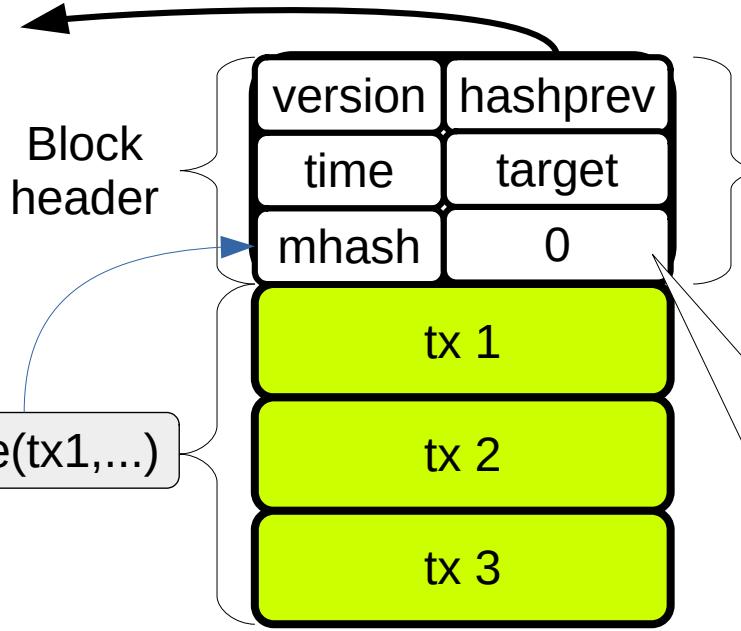
32 bit value can be
Changed to find PoW
via brute-force search

[*] Actually this is called *bits* or *nBits* and requires a conversion to get the target value

[**] Bitcoin uses SHA256(SHA256(b))

Mining

Block n - 1



No luck on first try

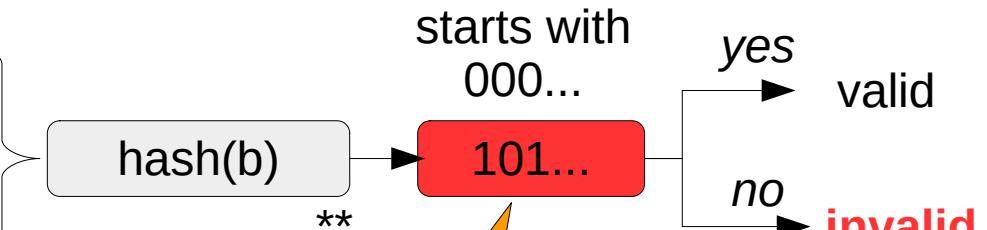
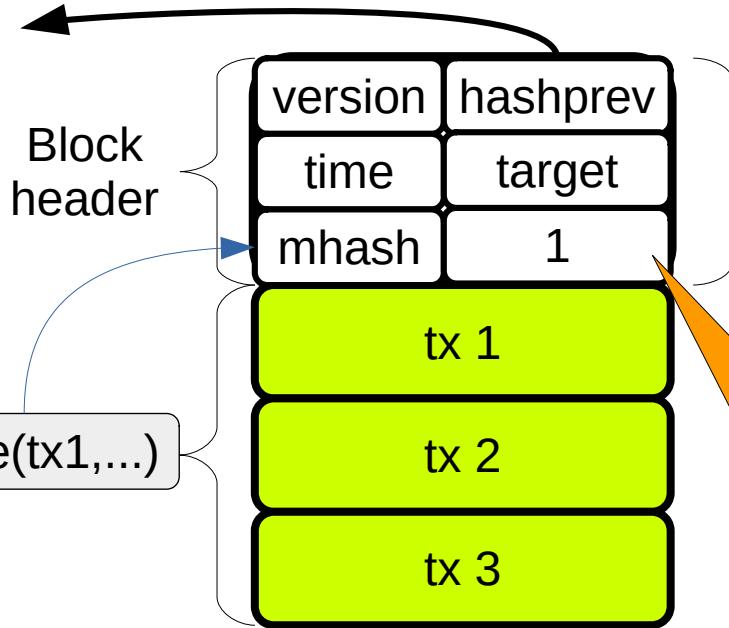
32 bit value can be
Changed to find PoW
via brute-force search

[*] Actually this is called *bits* or *nBits* and requires a conversion to get the target value

[**] Bitcoin uses SHA256(SHA256(b))

Mining

Block n - 1



No luck on second try

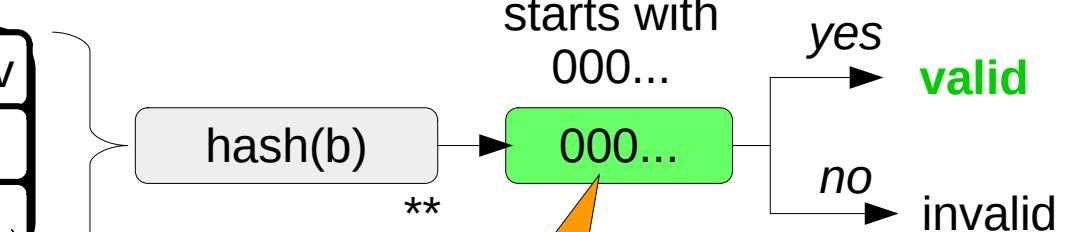
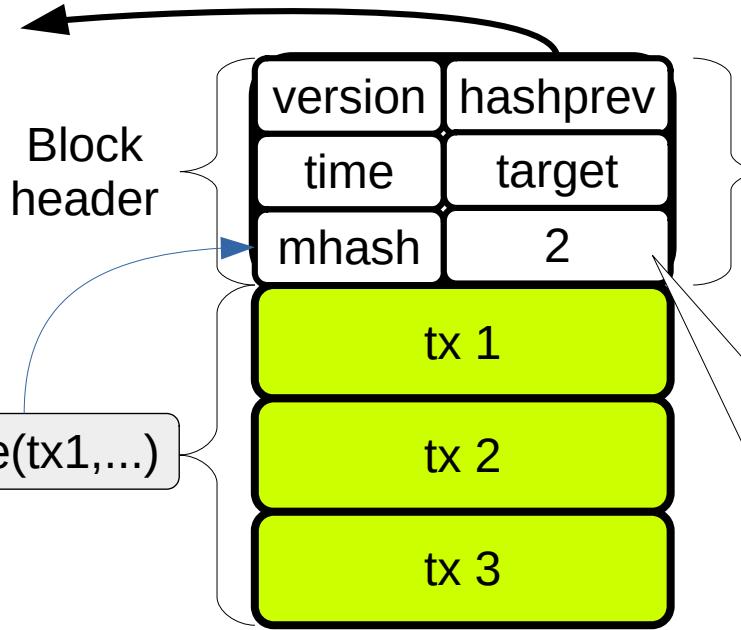
Increment nonce and try until we have a valid PoW

[*] Actually this is called *bits* or *nBits* and requires a conversion to get the target value

[**] Bitcoin uses SHA256(SHA256(b))

Mining

Block n - 1



Valid PoW on
3rd try

32 bit value can be
Changed to find PoW
via brute-force search

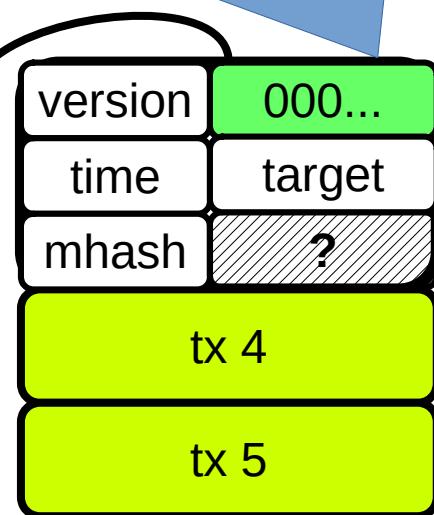
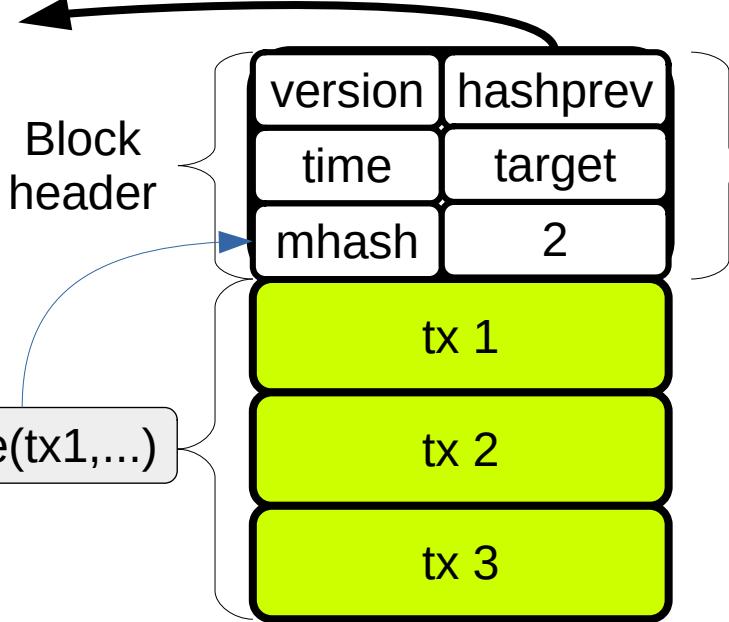
[*] Actually this is called *bits* or *nBits* and requires a conversion to get the target value

[**] Bitcoin uses SHA256(SHA256(b))

Mining

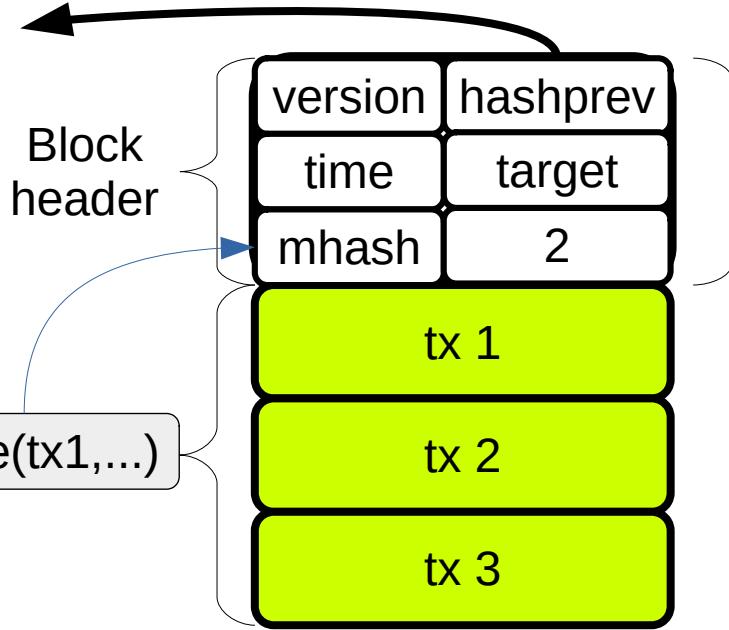
Miners build on top of another Valid block by referencing it

Block n - 1



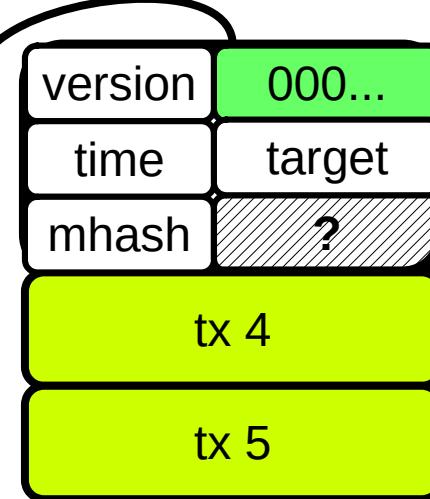
Mining

Block n - 1



Block n

1 confirmation on Block
2 confirmations for Transactions



Block n + 1

Only confirmed if another Block builds on top

0 confirmations on Block
1 confirmation for Transactions

Tx Confirmed if valid PoW

Mining - Success rate

- As a miner you are interested how long it takes to solve this puzzle
- This depends on your **fraction of the hash rate** and the **block interval**

$$\text{Mean time to next block} = \frac{10\text{min}}{\text{fraction of overall hash rate}}$$

- e.g. you have 0.001 % of the total network hash power, you're going to find blocks once every 7 days

Difficulty & Target

- **Target** defines the size of set for valid PoW solutions
 - To be valid a hash value has to be smaller than the current target
 - The maximum target value is MAX_TARGET = $2^{224} = 0x00000000FFFFFFFFFFFFFFF...FF$
 - i.e. 256 bit value with 32 leading zero bits
 - on average 2^{32} attempts puzzle until solved
- **Difficulty** is the ratio of highest possible target and current target

$$\text{difficulty} = \frac{T_{max}}{T_{current}}$$

- When target is *large*, the difficulty is *low*
- When target is *small*, the difficulty is *high*

Difficulty & Target

- In Bitcoin **target** is readjusted every 2016 blocks
 - Rule of thumb:
one block should be found every 10 min
 - Readjust target (~14 days) to match that:

$$T = T_{previous} * \frac{t_{actual}}{2016*10min}$$

t_{actual} = time passed since last block

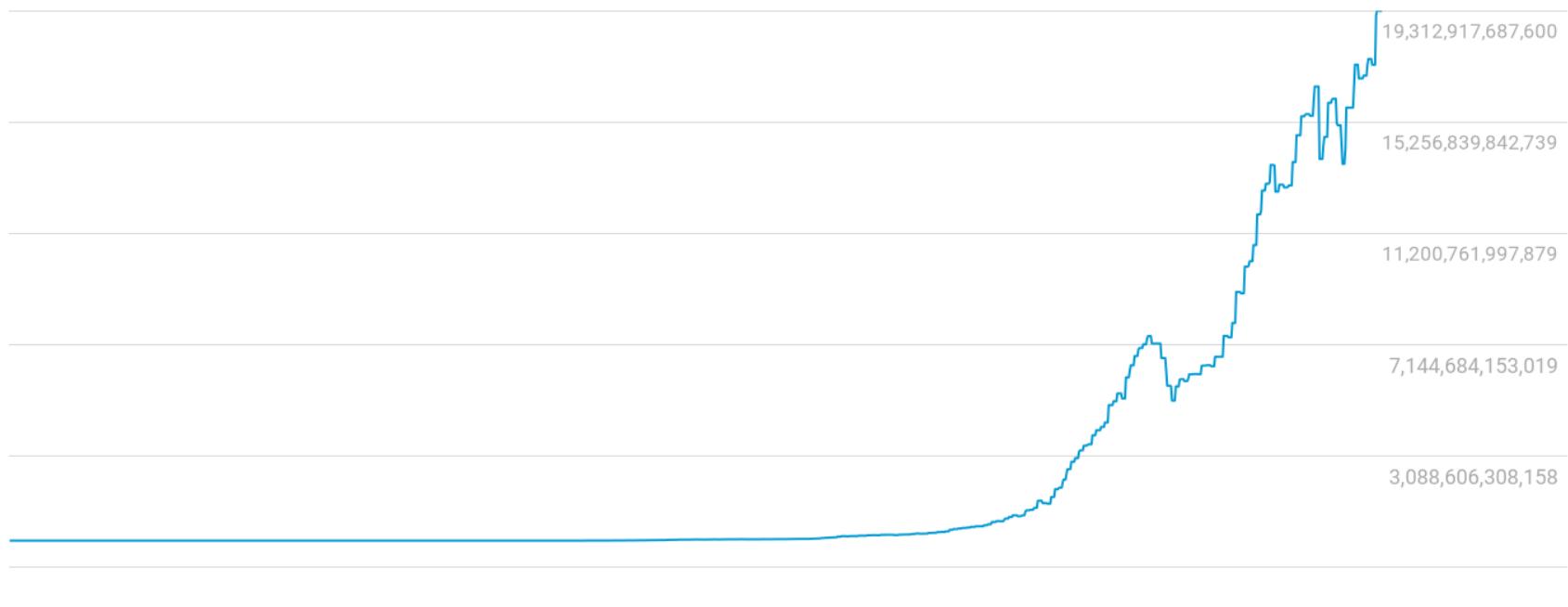
Note: bitcoin-core actually has an off-by-one error and only considers 2015 blocks in the calculation

Difficulty & Target

- Bitcoin readjusts hardness automatically

Difficulty

19,298,087,186,263



Hash rate



CPU



GPU



FPGA



ASIC



gold pan



sluice box

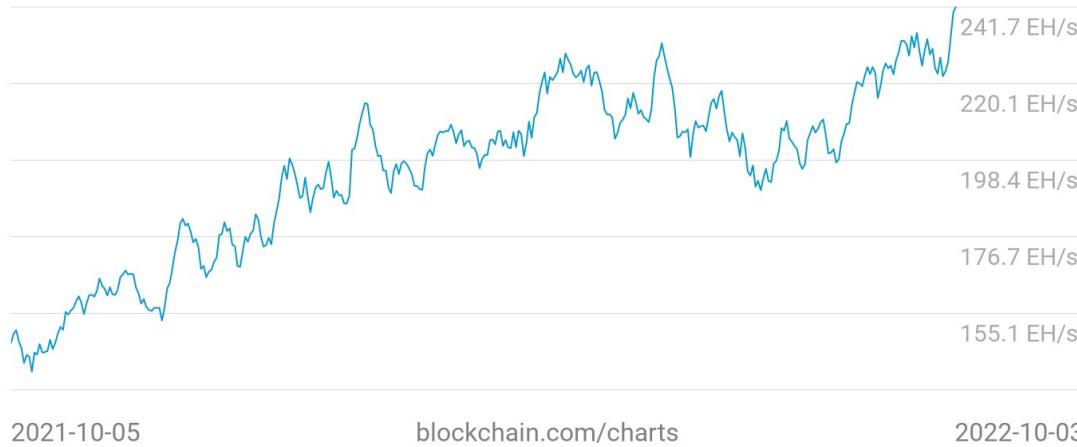


placer mining

pit mining

Hash Rate

241.7 EH/s



Hash rate

Currently at ~ 240,000,000,000 GH/s = 240,000,000 TH/s = 240,000 PH/s
~ 240 EH/s

For scale:

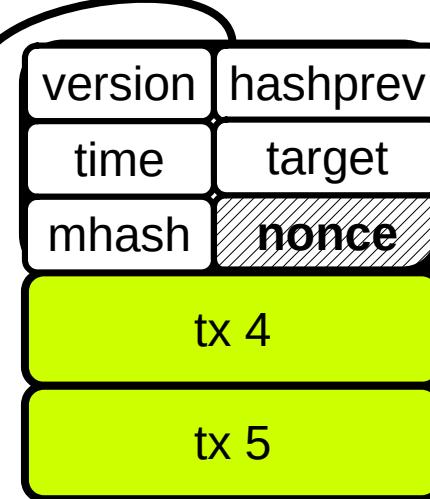
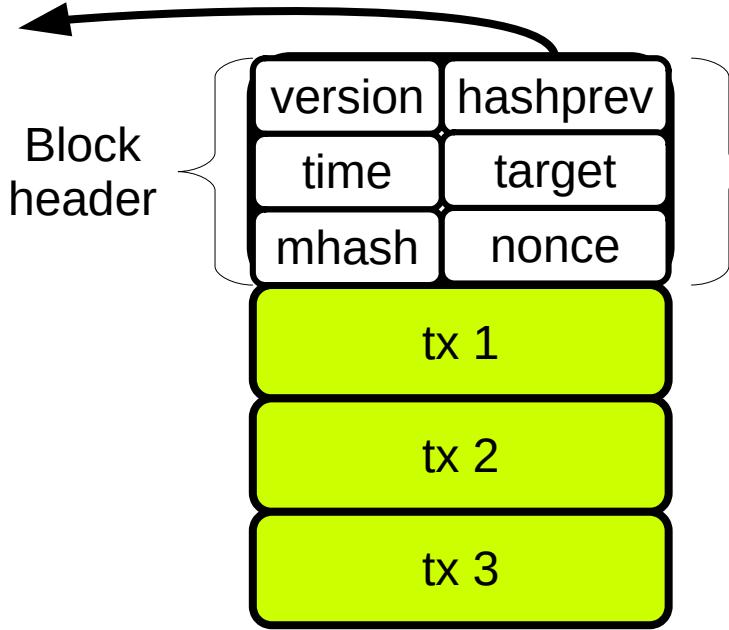
- one ASIC miner delivers ~1000 GH/s to 13.5 TH/s
- one FPGA miner delivers ~ 1 GH/s
- one GPU miner delivers ~ 0.1 GH/s
- one CPU miner delivers ~ 0.01 GH/s

[1] <https://blockchain.info/charts/hash-rate>

[2] https://en.bitcoin.it/wiki/Mining_Hardware_Comparison

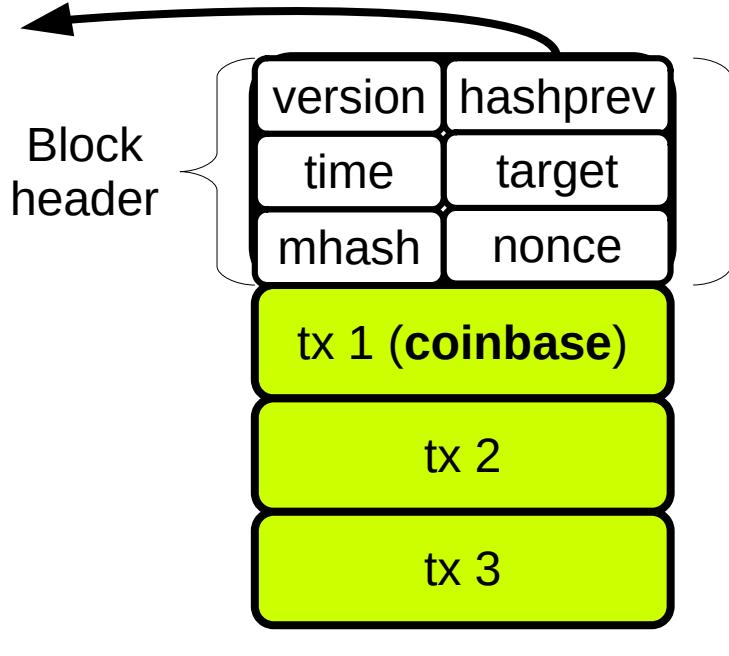
Mining

Block n - 1

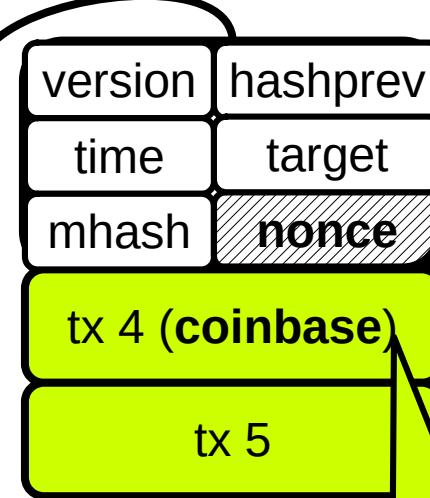


Mining

Block n - 1

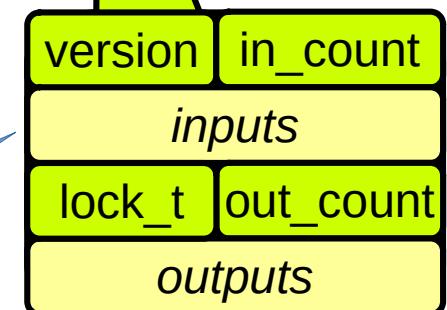


Block n



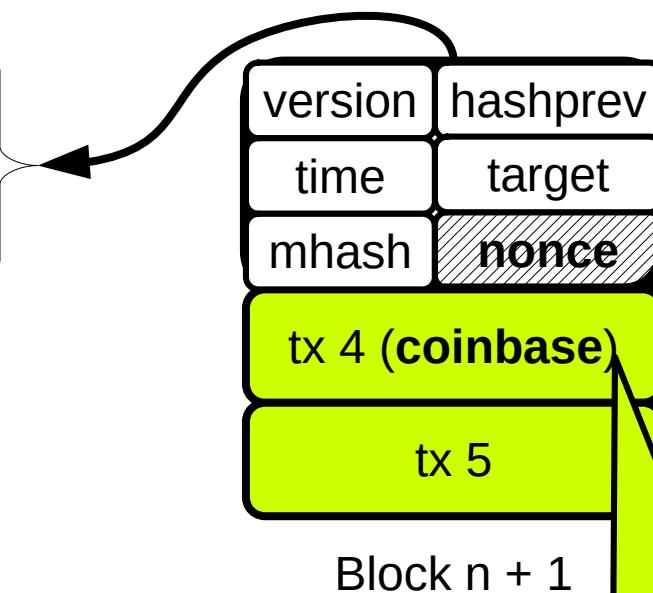
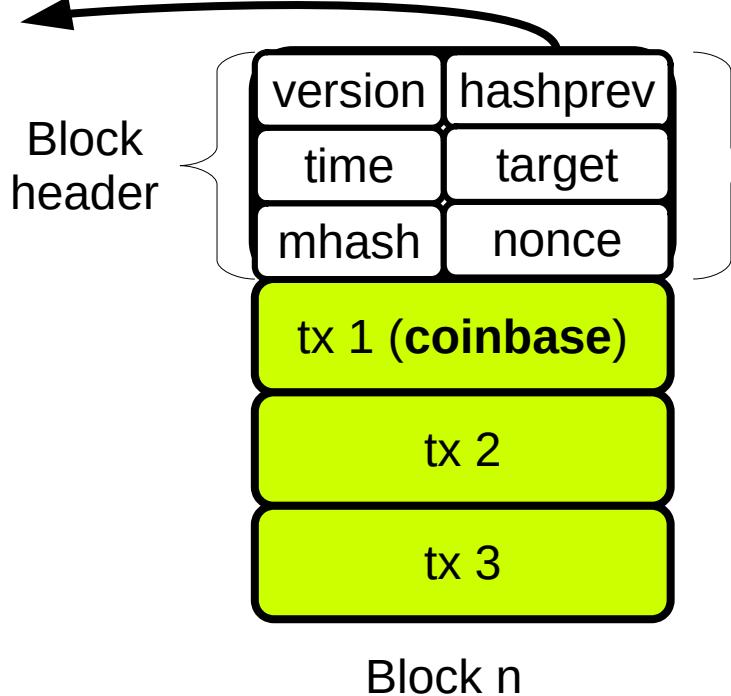
Block n + 1

A transaction is made up of **inputs** and **outputs**

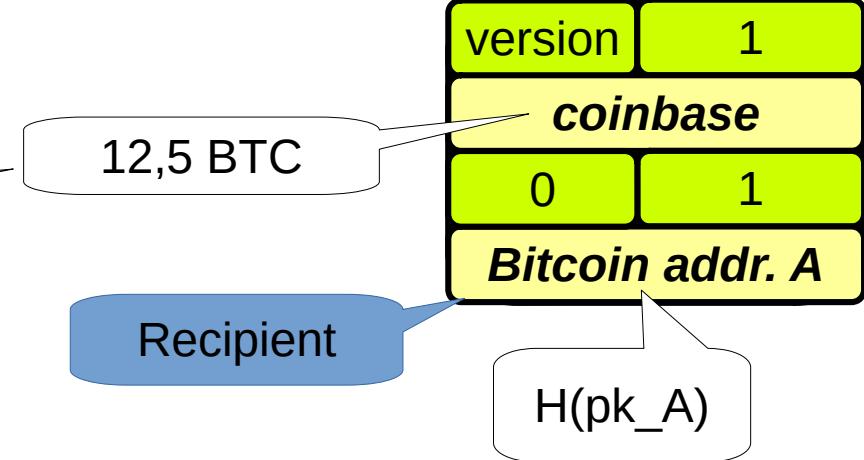


Mining

Block n - 1

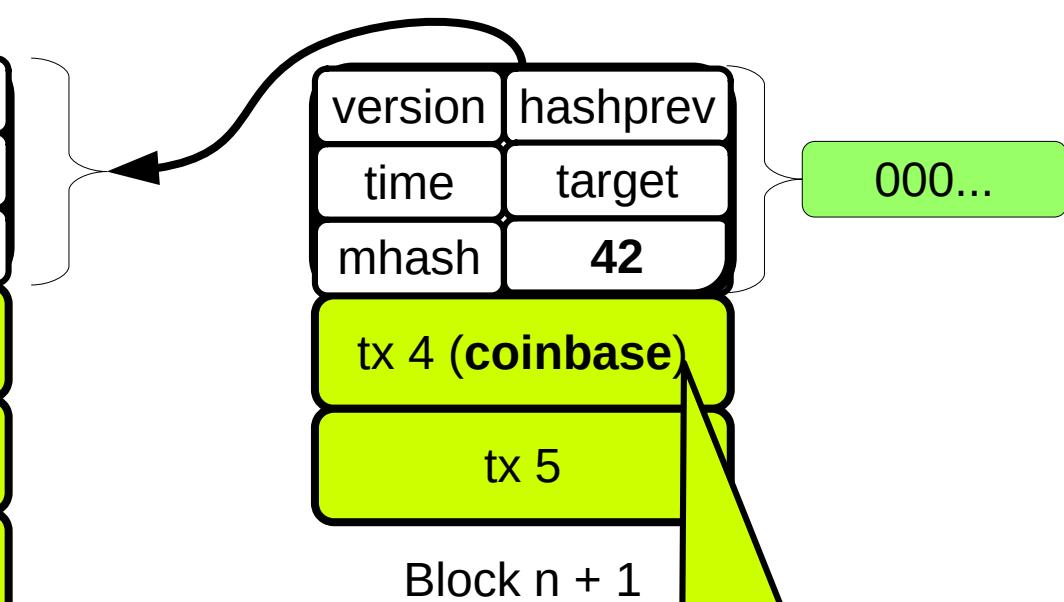
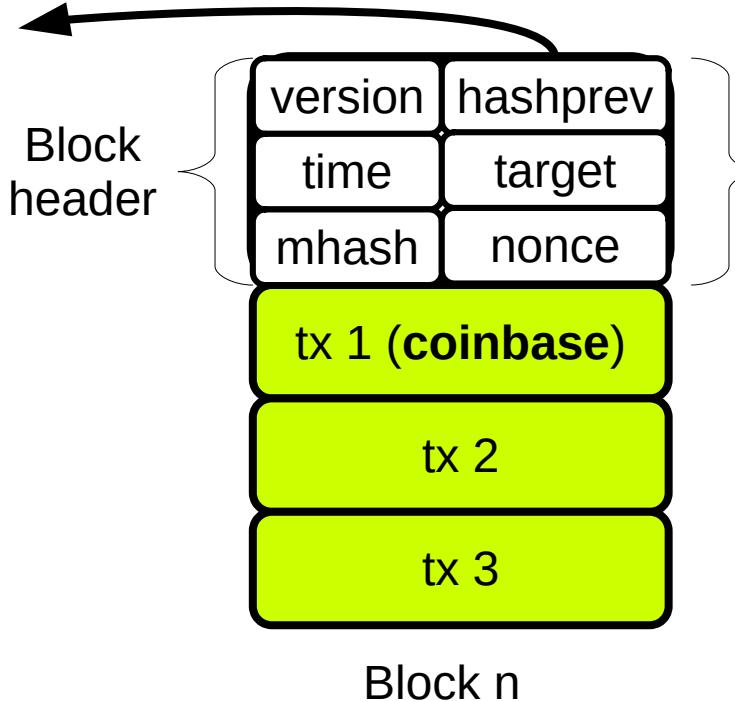


The **coinbase** is where **new bitcoins** come into existence



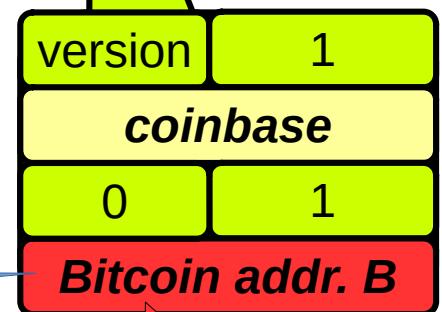
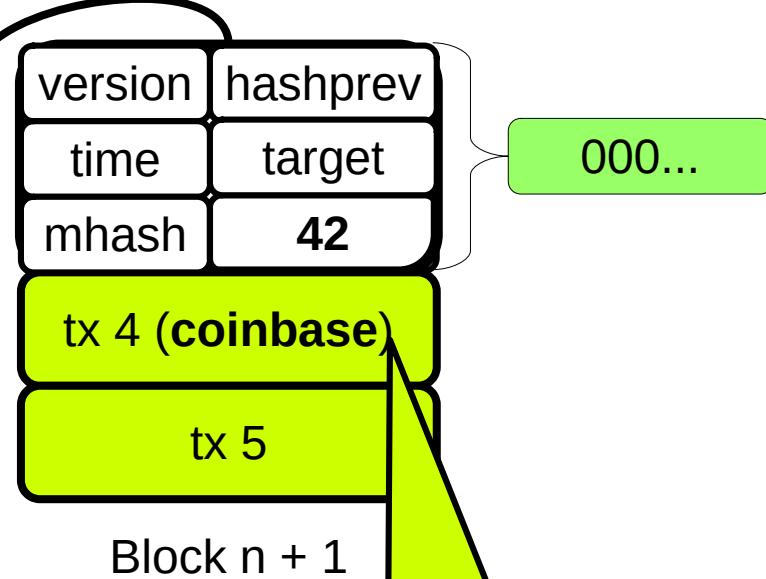
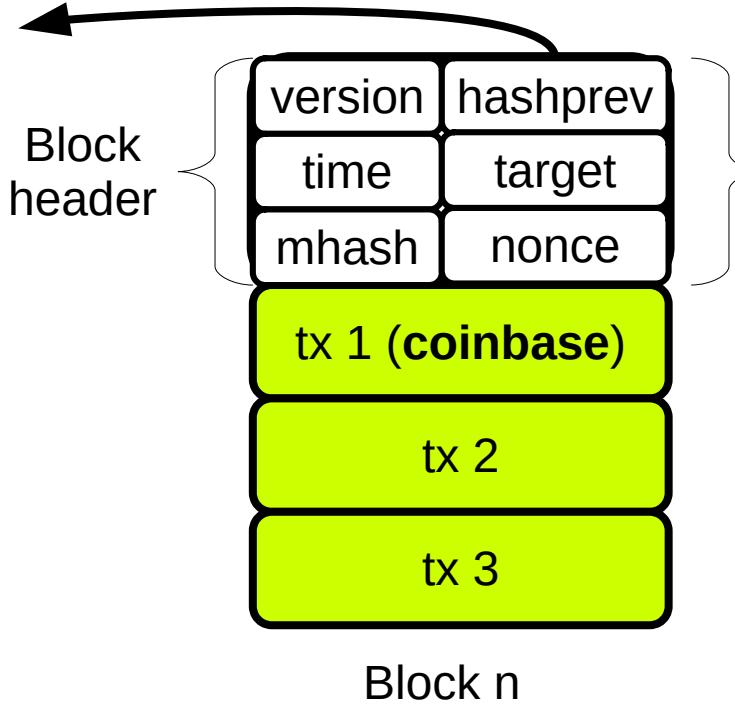
Mining

Block n - 1



Mining

Block n - 1

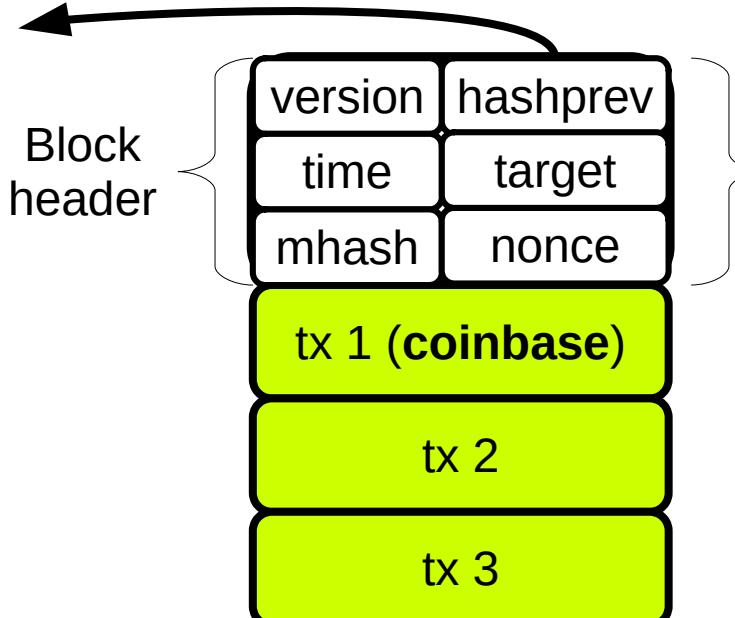


What if someone simply changes the recipient?

$H(pk_M)$

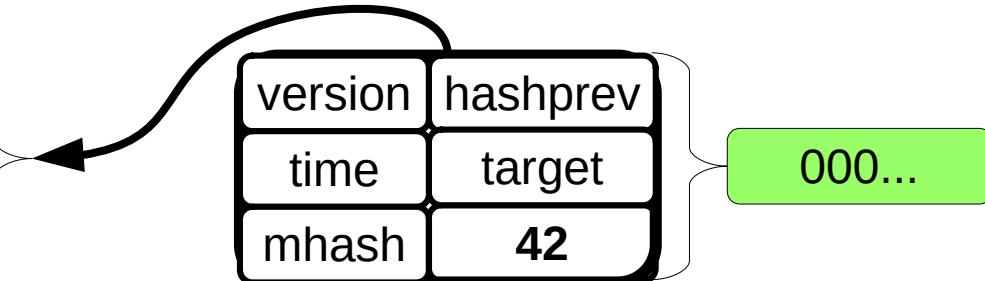
Mining

Block n - 1

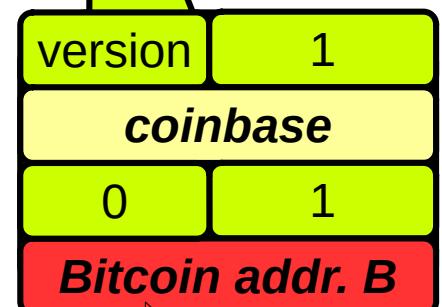


Block n

The transaction data will change



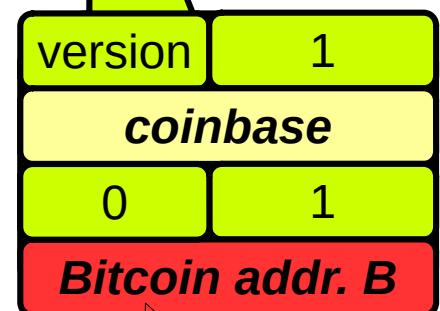
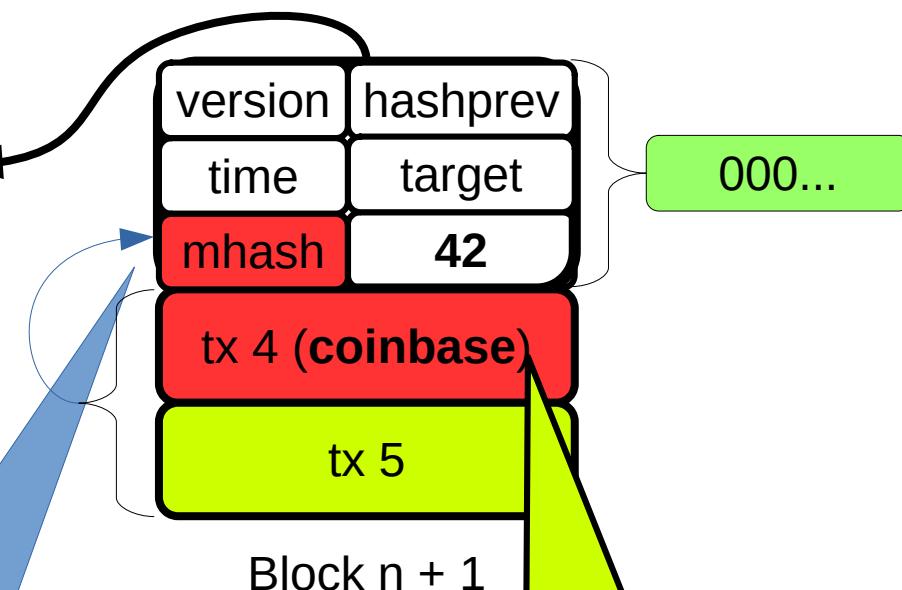
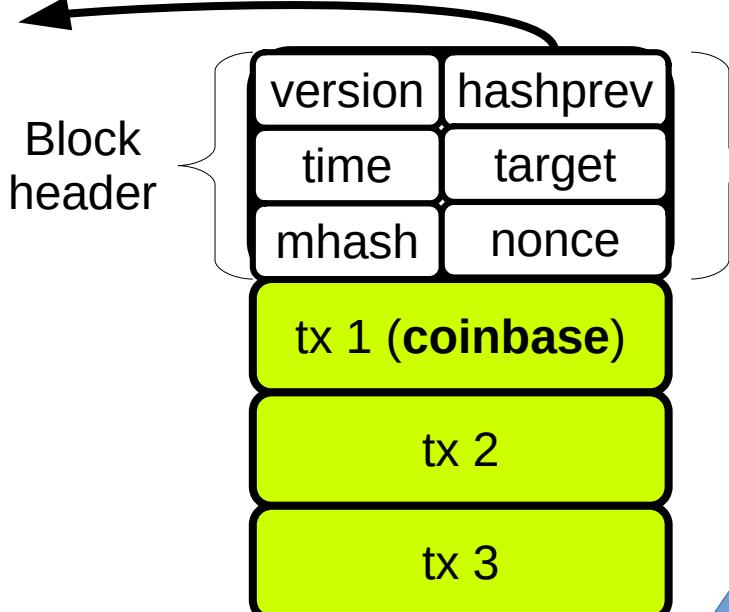
Block n + 1



$H(pk_M)$

Mining

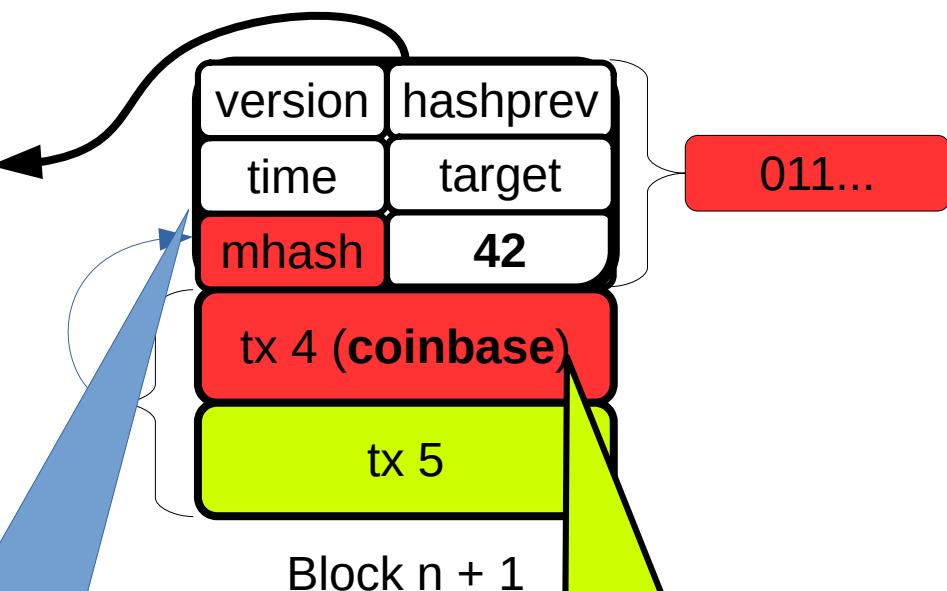
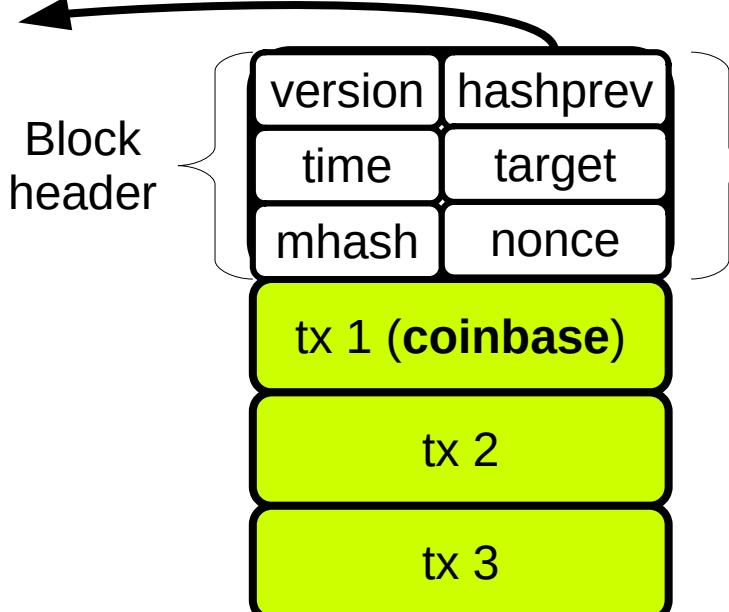
Block n - 1



Recall: Small changes in data still have a big impact on their hash

Mining

Block n - 1



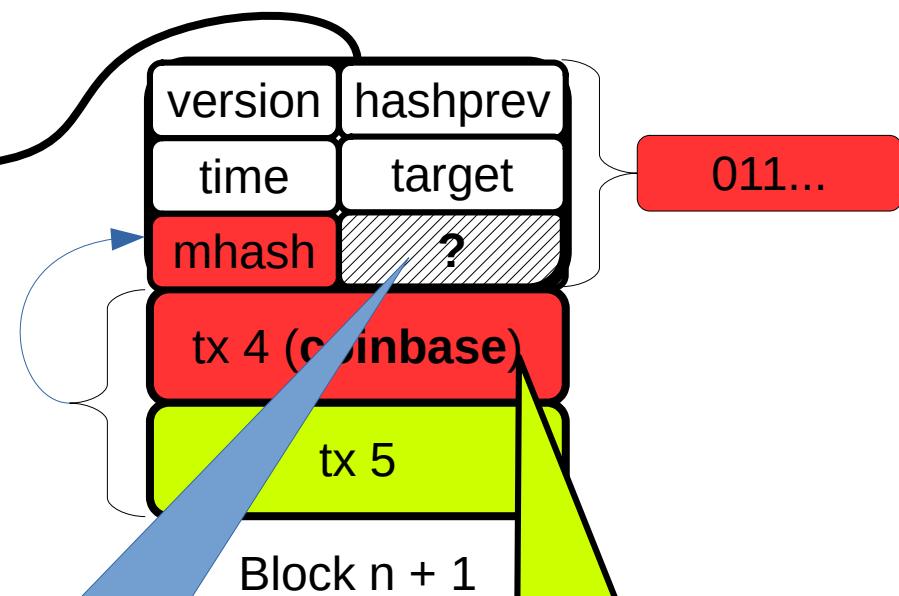
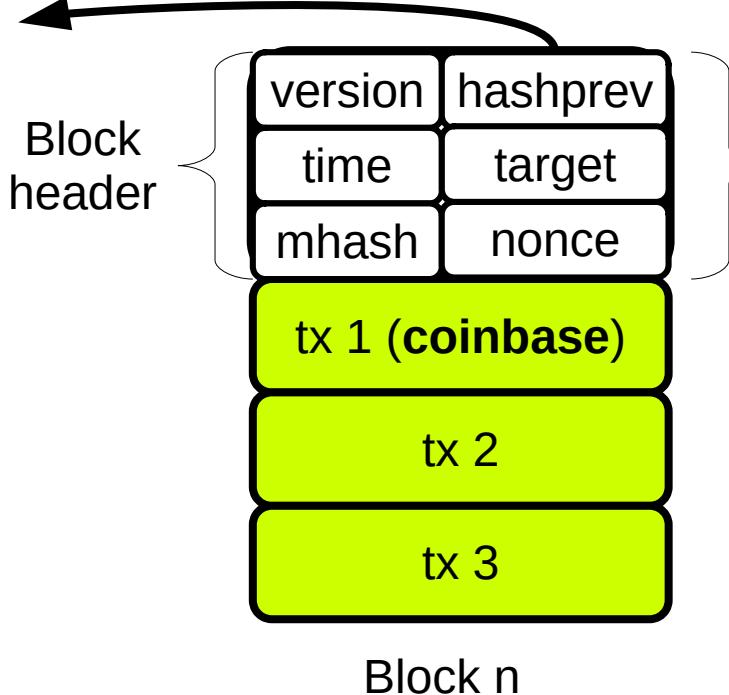
Bitcoin addr. B

H(pk_M)

Any changes to the Block header
will change its hash, therefore
Invalidating the PoW

Mining

Block n - 1



$H(pk_M)$

To make this a valid Block
the attacker will have to re-do
the PoW...

Bitcoin address

- A **Bitcoin address** is a base-58 encoded hash* of a public key e.g.

1Ak7KcXC571HMLhxJ6ZqFr7CxKyuGWxsr

[*] Actually it is:

```
Base58(0x00 | RIPEMD160( SHA256(pk) ) ||  
        ${SHA256( SHA256(  
            0x00 | RIPEMD160( SHA256(pk) ) ))  
            :0:4 })
```

Bitcoin address

pub. key

[*] Actually it is:

```
Base58 (0x00 | RIPEMD160 ( SHA256(pk) ) ||  
         ${SHA256( SHA256(  
         0x00 | RIPEMD160 ( SHA256(pk) ) ))  
         :0:4 }  
)
```

4 byte

“checksum”

Hex / Leading Symbol / Usage

0x00 = 1... = Pay-to-pubkey-hash (p2pkh)

0x05 = 3... = Pay-to-script-hash (p2sh)

0x80 = 5... = Private key

0x6f = m/n = Testnet p2pkh

0xc4 = 2... = Testnet p2sh

Bitcoin address

- Why base-58

```
/**  
 * Why base-58 instead of standard base-64 encoding?  
 * - Don't want 00IL characters that look the same  
 *   in some fonts and could be used to  
 *   create visually identical looking data.  
 * - A string with non-alphanumeric characters is  
 *   not as easily accepted as input.  
 * - E-mail usually won't line-break if there's  
 *   no punctuation to break at.  
 * - Double-clicking selects the whole string as  
 *   one word if it's all alphanumeric.  
 */
```

Bitcoin address

- Since Bitcoin addresses are basically **public/private key pairs** one can prove ownership by signing a message with the corresponding private key

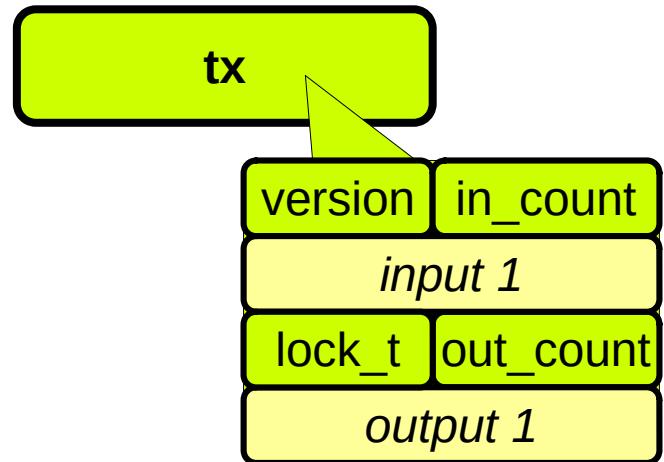
```
$ ./bitcoin-cli signmessage 12yeU5ymM67SL5UWVSwErAgwVwwaTd1Nma \
"https://www.soscisurvey.de/BTC_study/"
HzzNxFmeRhbhAwVZ4DsraBkXkW7JYj00tAlIPAnHB2z5P12eddFilWXJmwGm\
PkgS/v8W0DNr0Z1qLwroPbWWMoE=
```

Bitcoin address

- Can be created by anybody
 - e.g. <https://www.bitaddress.org>
- Enough to receive bitcoins without knowing anything about the protocol or having to install a client

Transactions

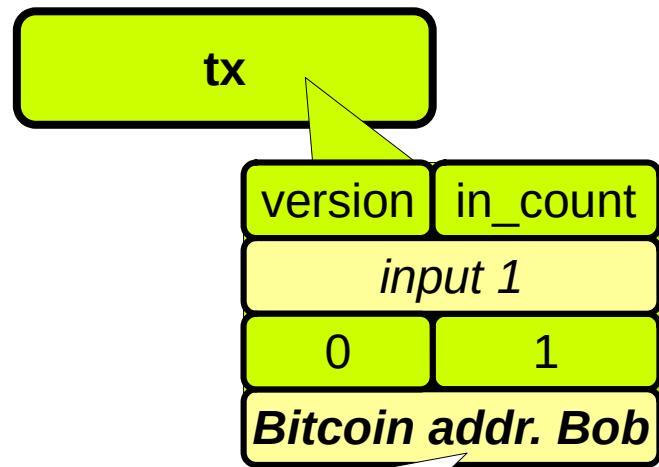
- Alice wants to send bitcoins to Bob



Transactions

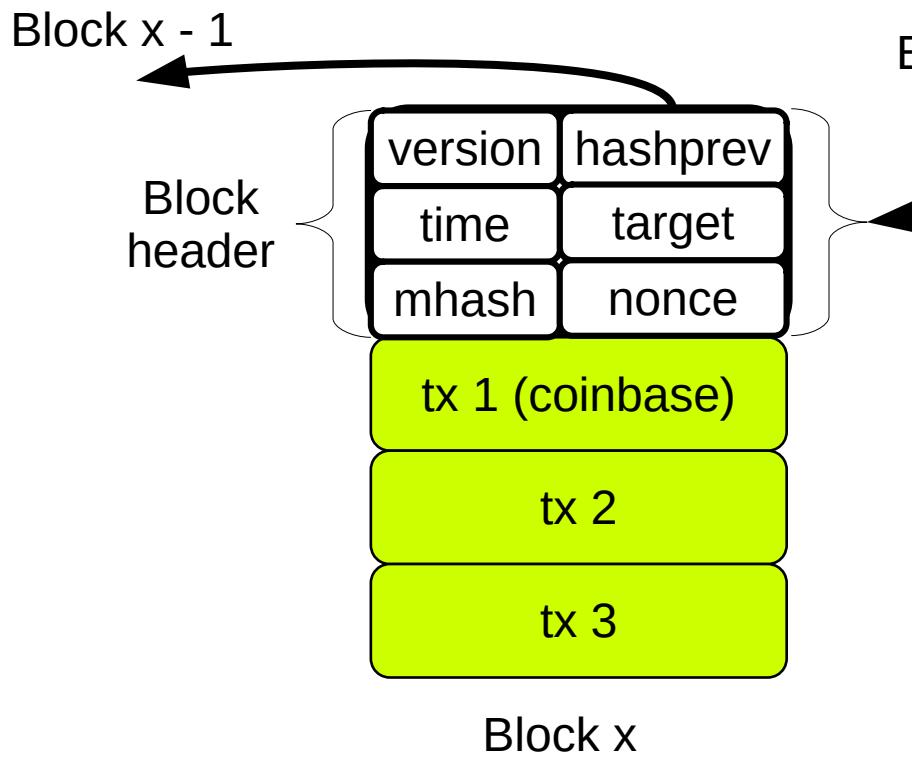
- Alice wants to send bitcoins to Bob

Create a tx where recipient Address (**output**) is hash of Bob's public key

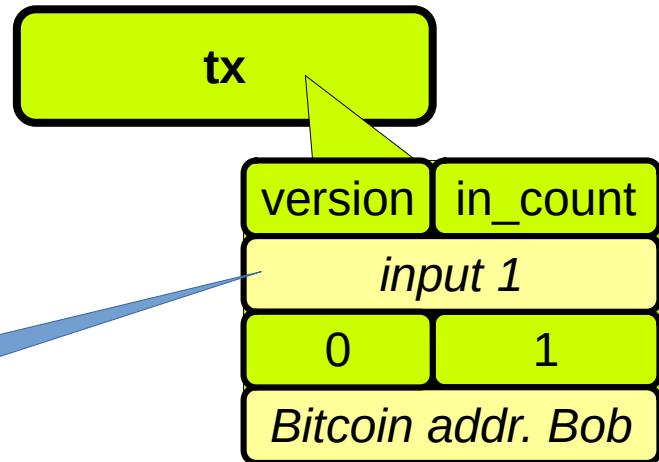


$H(pk_B) =$
1MuSWqHzrtHmSGHEuvj3N41SWv1dpf9zQV

Transactions

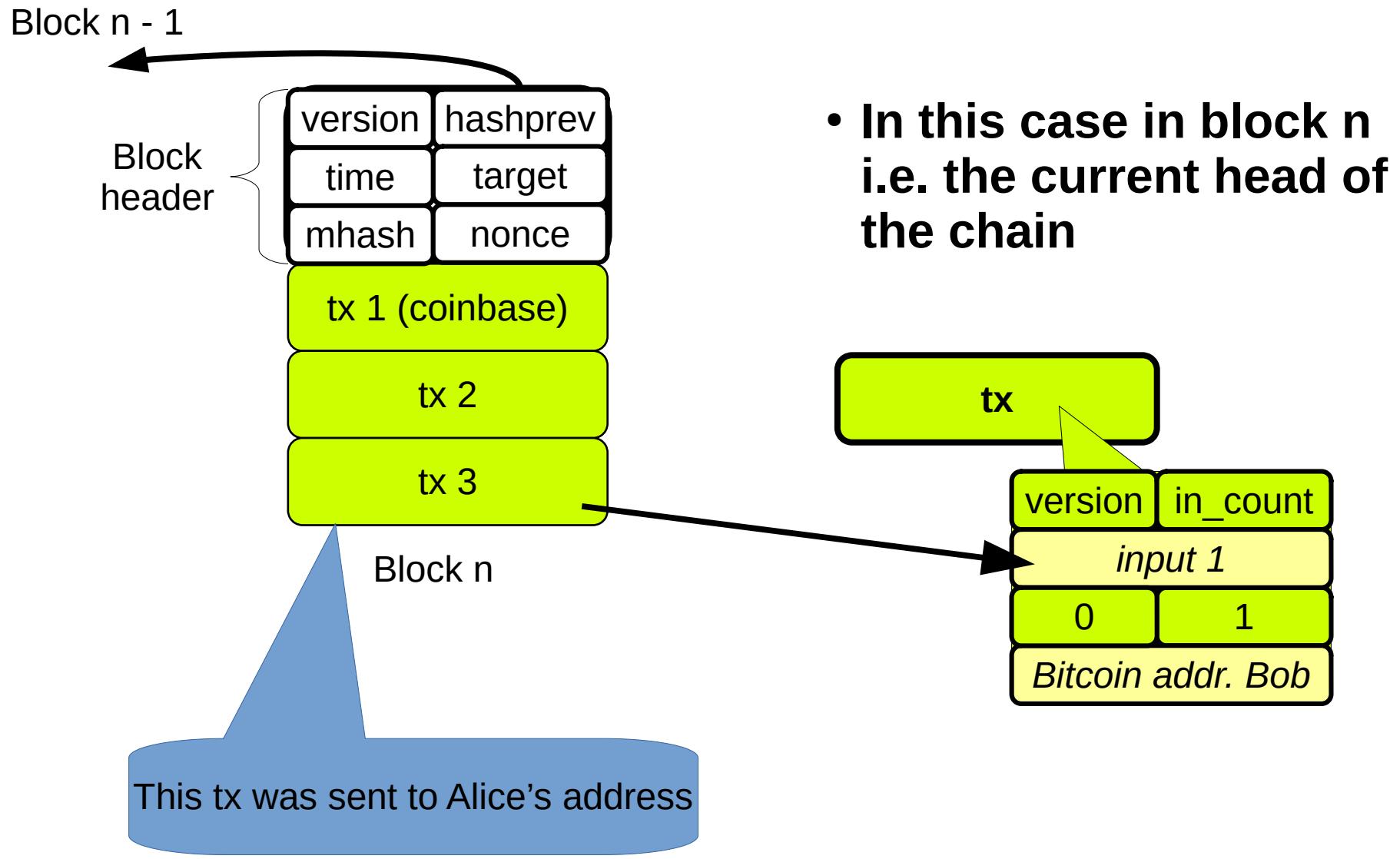


- **Search block with tx that has an output to Alice**
- **Can be anywhere in the blockchain**



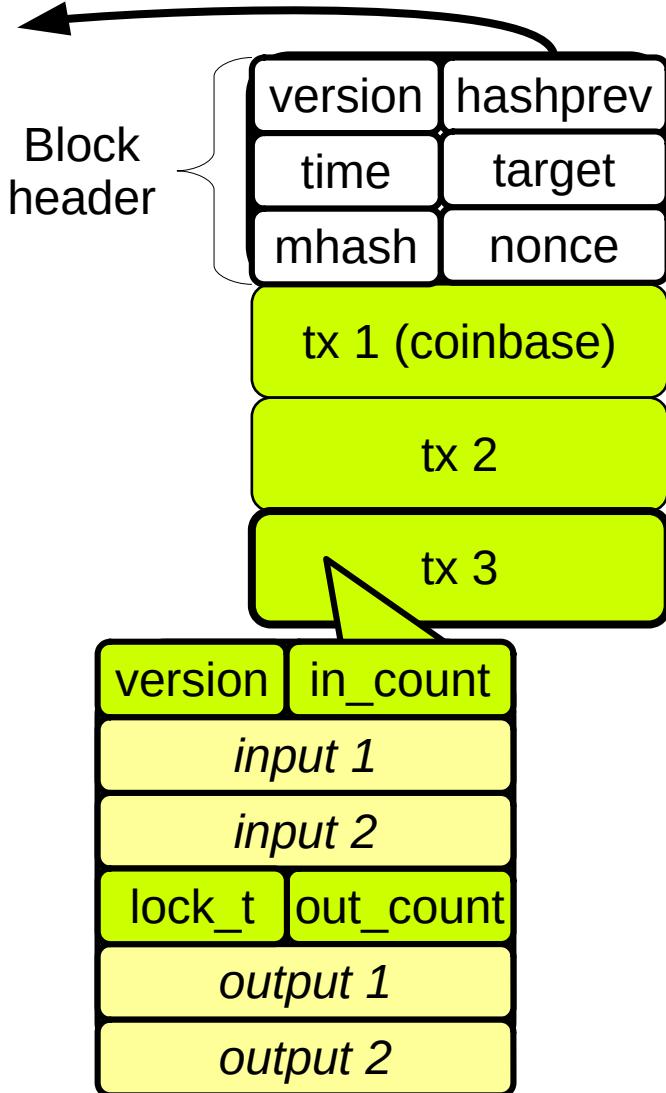
As a valid **input** Alice needs a tx that was sent to her and which she has not spent yet

Transactions

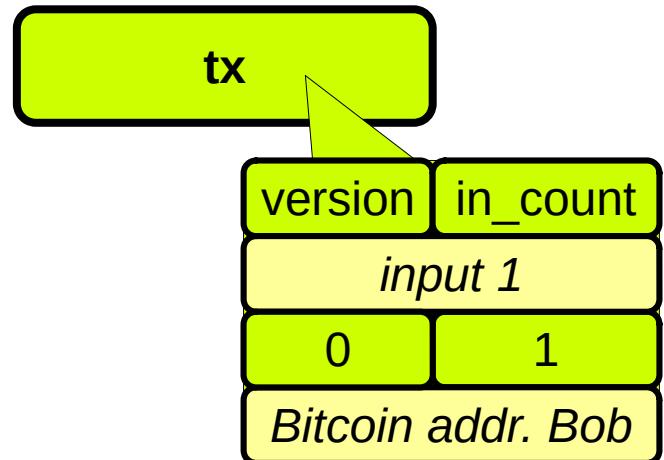


Transactions

Block n - 1

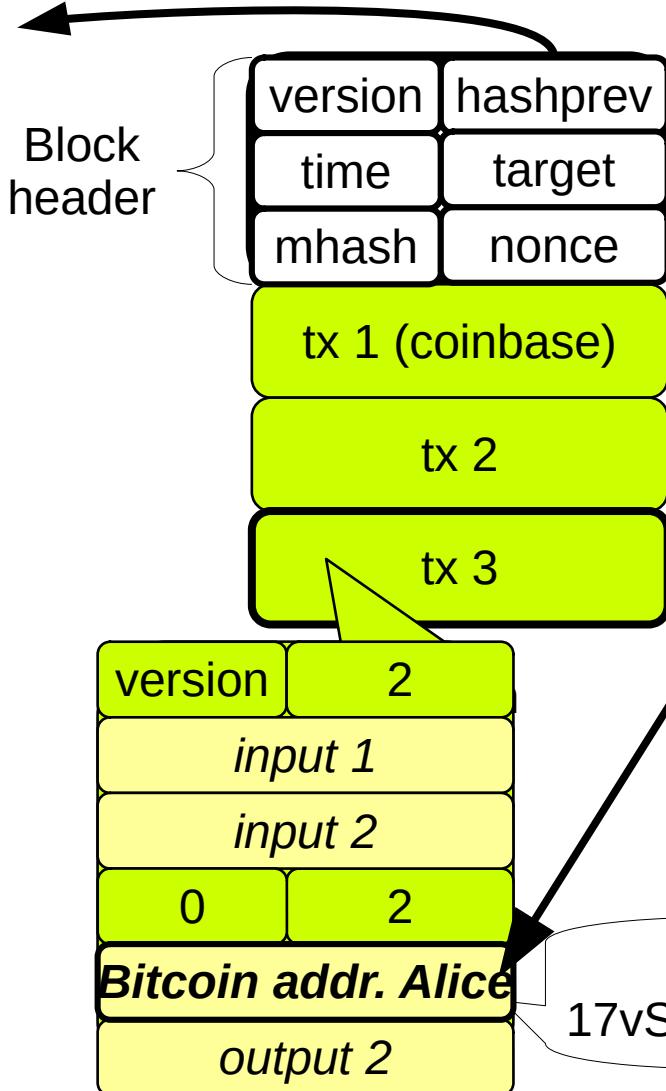


Alice needs to **unlock** this previous output



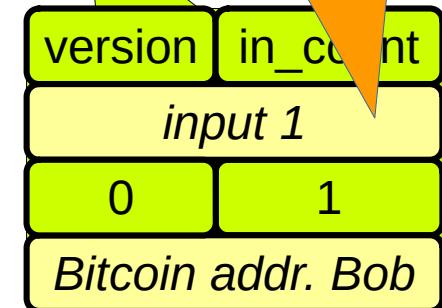
Transactions

Block n - 1



Create a signature with Alice's private key to **unlock** received funds

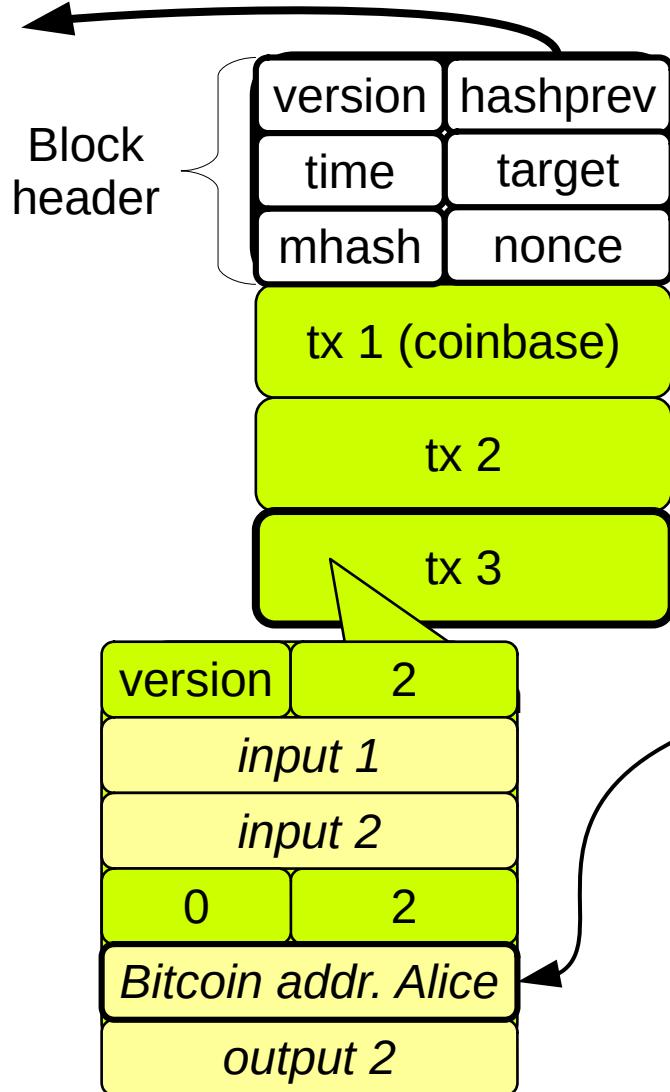
tx



$H(pk_A) =$
17vS2zLBnSDc2URGLXu7eqmvYNAdJ9m6cA

Transactions

Block n - 1

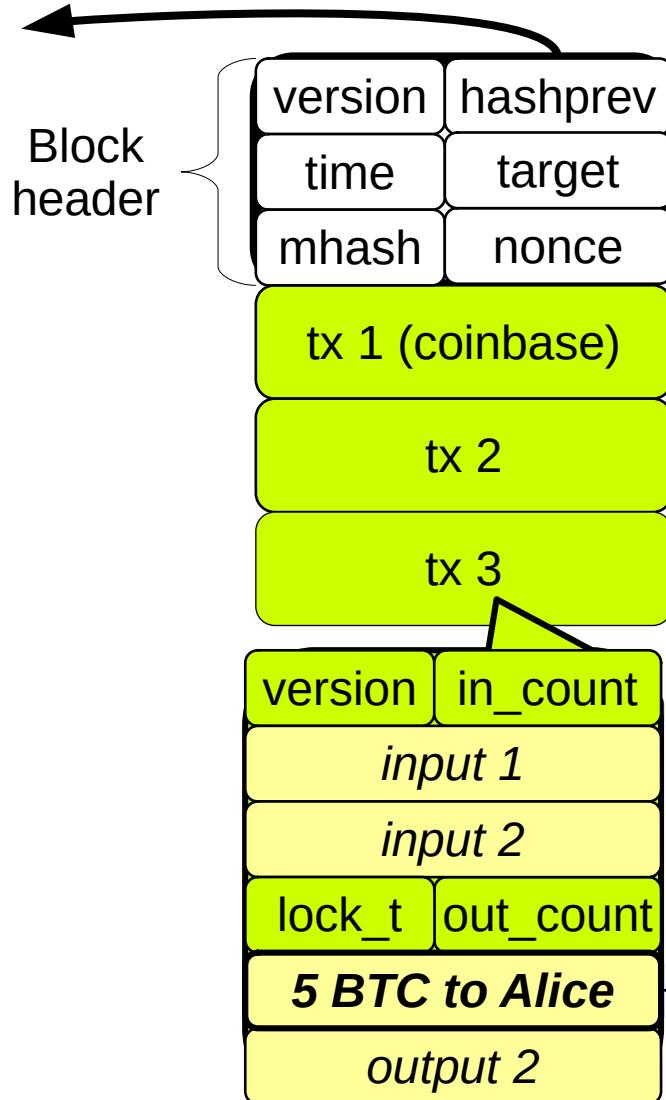


Alice references the **output of a previous transaction** and signs the new **transaction** with her **secret key**

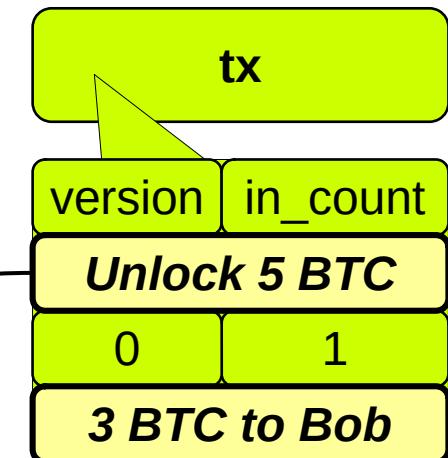
version	1
$sign(sk_A, tx); pk_A$	
0	1
Bitcoin addr. Bob	

Transactions - Change

Block n - 1

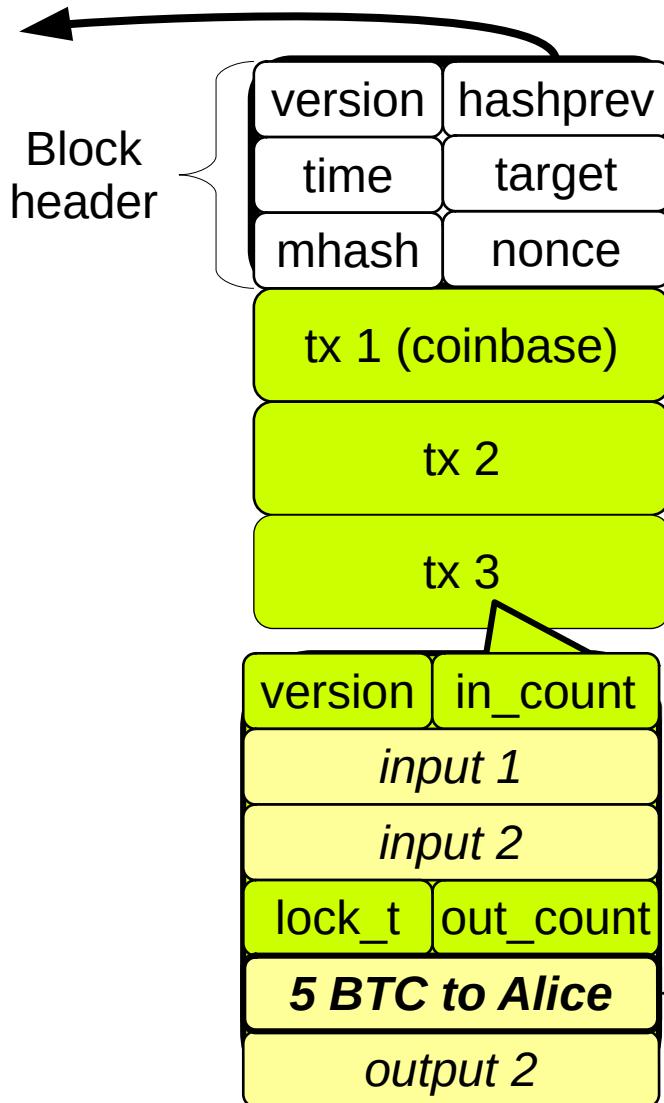


What happens to the remaining 2 BTC?

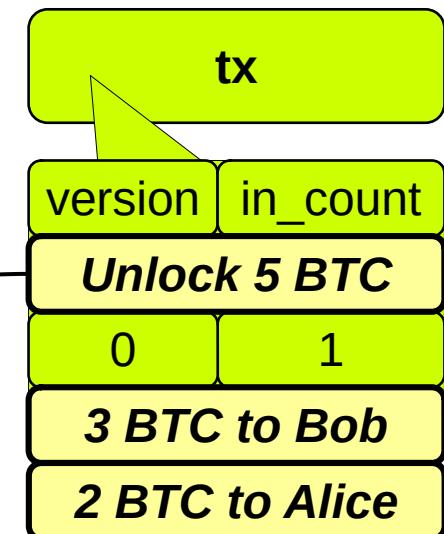


Transactions - Change

Block n - 1

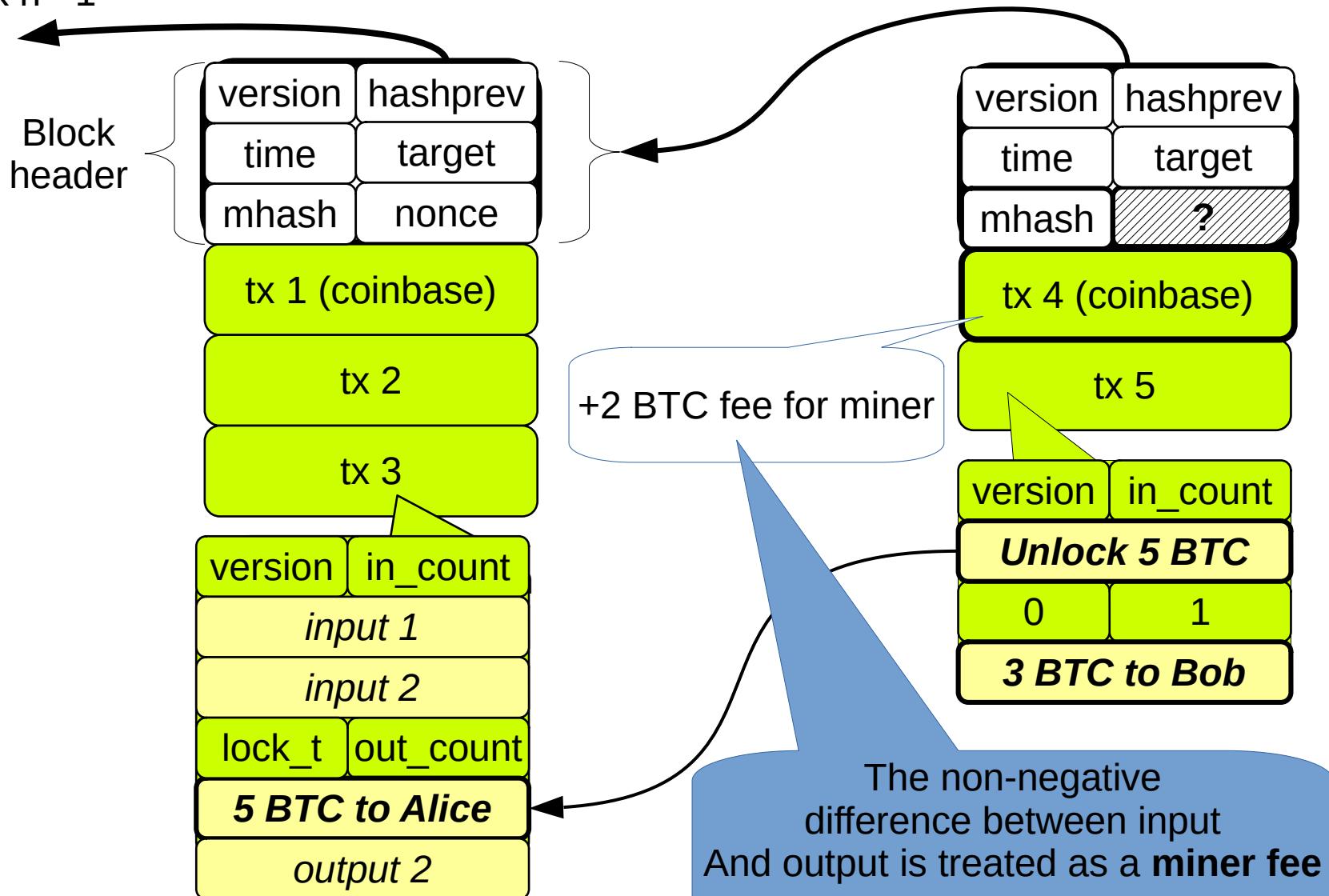


Alice sends the remainder to one of her own addresses



Transactions - Fee

Block n - 1



Transactions – Scripting Language

- Bitcoin has its own scripting language for transactions
- Actually inputs and outputs are **scripts** in a *stack based, non-Turing complete* language
 - No loops
- A script is *only valid* if it **executes successfully** with no errors otherwise the transaction is not valid and will not be included

Transactions – Scripting Language

- A transaction ***output*** doesn't just specify a public key but a script saying for example
 - “*this output can be redeemed by a public key that hashes to X, along with a signature from the owner of that public key.*”
 - **ScriptPubKey**
- The according transaction ***input*** then needs to specify a redeem script which is saying
 - “*Here is the public key and the signature you are looking for*”
 - **ScriptSig**
- Both scripts are simply concatenated (the input and the earlier output script) and the resulting script must run successfully in order for the transaction to be valid.

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

~~<sig>~~ <pubKey> OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

<sig>

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

~~<sig> <pubKey>~~ OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

<pubKey>
<sig>

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

~~<sig> <pubKey>~~ OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

<pubKey>
<pubKey>
<sig>

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

~~<sig> <pubKey> OP_DUP OP_HASH160~~ <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

<pubKeyHash>
<pubKey>
<sig>

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

<~~sig~~> <~~pubKey~~> OP_DUP OP_HASH160 <~~pubKeyHash~~>
OP_EQUALVERIFY OP_CHECKSIG

<pubKeyHash>
<pubKeyHash>
<pubKey>
<sig>

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

<~~sig~~> <~~pubKey~~> OP_DUP OP_HASH160 <~~pubKeyHash~~>
~~OP_EQUALVERIFY~~ OP_CHECKSIG

<pubKey>

<sig>

Transactions – Scripting Language

- Evaluation *Pay-to-PublicKeyHash* (P2PKH) [1]:

scriptPubKey (locks output):

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig (unlocks output i.e. in input):

<sig> <pubKey>

~~<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>~~
~~OP_EQUALVERIFY OP_CHECKSIG~~

true

Transactions – Scripting Language

- It is generally planned to add more functionality (operands) to Bitcoin Script in the future

Transactions – Scripting Language

- The rationale behind using Script

satoshi
Founder
Sr. Member


Activity: 364



Ignore



**Re: Transactions and Scripts: DUP HASH160 ... EQUALVERIFY
CHECKSIG**

June 17, 2010, 06:46:08 PM

quote #2

The nature of Bitcoin is such that once version 0.1 was released, the core design was set in stone for the rest of its lifetime. Because of that, I wanted to design it to support every possible transaction type I could think of. The problem was, each thing required special support code and data fields whether it was used or not, and only covered one special case at a time. It would have been an explosion of special cases. The solution was script, which generalizes the problem so transacting parties can describe their transaction as a predicate that the node network evaluates. The nodes only need to understand the transaction to the extent of evaluating whether the sender's conditions are met.

The script is actually a predicate. It's just an equation that evaluates to true or false. Predicate is a long and unfamiliar word so I called it script.

The receiver of a payment does a template match on the script. Currently, receivers only accept two templates: direct payment and bitcoin address. Future versions can add templates for more transaction types and nodes running that version or higher will be able to receive them. All versions of nodes in the network can verify and process any new transactions into blocks. even though they may not know how to read them.

[1] <https://bitcointalk.org/index.php?topic=195.msg1611#msg1611>

Blockchain

Definition & Types

What is a “Blockchain”

- Multiple definitions:
 - **Umbrella term**
“something kind of like Bitcoin”
 - **Data structure**
 - NIST
“the actual ledger”
 - Princeton definition [1]
“*Linked list with hash pointers instead of pointers*”
 - **Distributed system**
 - (Permissionless) *Nakamoto Consensus*
 - Abstract definition [2,3]
 - Common prefix
 - Chain quality
 - Chain growth



[1] <https://allquantor.at/blockchainbib/#narayanan2016bitcoin>

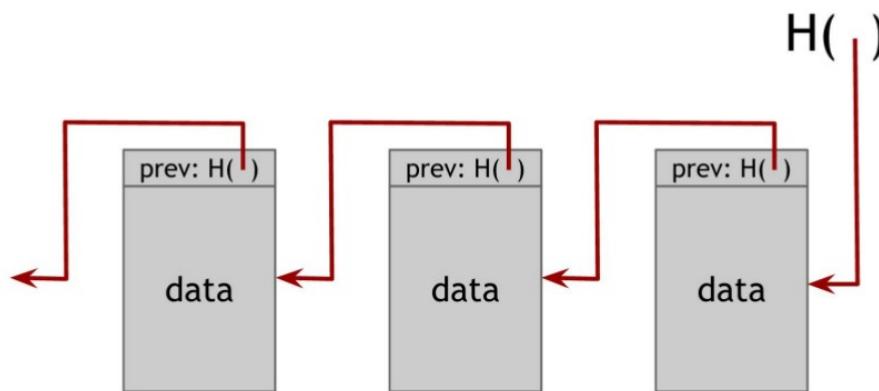
[2] <https://allquantor.at/blockchainbib/#garay2015bitcoin>

[3] <https://allquantor.at/blockchainbib/#pass2016analysis>

Blockchain -“Princeton” definition

“A block chain is a **linked list** that is built with **hash pointers** instead of pointers” [1]

- Digest of previous block included in every block
- as long as the hash pointer at the head of the list cannot be changed by the adversary, the adversary will be unable to change any block without being detected



Blockchain -“Princeton” definition

Pro:

- + Simple
- + Holds true for all types of blockchains, independent of used consensus system

Con:

- Only describes the ***offline data structure***
- Assumes head is safe, does not mention any consensus system

Blockchain -“Abstract” definition

“A blockchain is an **interactive protocol** where each participant has a **local variable state** which contains a *list of messages m*, called the “chain”.

Players receive inputs, called records/batches/messages, that they attempt to include in the chain of themselves and of others.” [1]

[1] [pass2016analysis]

Blockchain -“Abstract” definition

- The following properties are required:
 - **consistency**: with overwhelming probability (in T), at any point, the chains of two honest players can differ only in the last T blocks;
 - future **self-consistence**: with overwhelming probability (in T), at any two points r, s the chains of any honest user at r and s differs only in blocks within the last T blocks;
 - the **μ -chain quality** with overwhelming probability (in T), for any T consecutive messages in any chain held by some honest player, the fraction of messages that were “contributed by honest players” is at least μ .
 - **g -chain-growth**: with overwhelming probability (in T), at any point in the execution, the chain of honest players grew by at least T messages in the last T/g rounds, where g is called the chain-growth of the protocol.

Blockchain -“Abstract” definition

Backbone properties	Nakamoto BA protocol Π_{BA}^{nak}	Our BA protocol $\Pi_{BA}^{1/3}$	Public Ledger Π_{PL}	Our BA protocol $\Pi_{BA}^{1/2}$
common prefix	Agreement $\frac{1}{2}$	Agreement $\frac{1}{2}$	<i>Persistence:</i> transactions are permanent and ordered $\frac{1}{2}$	Agreement $\frac{1}{2}$
chain quality	Validity ϵ	Validity $\frac{1}{3}$	<i>Liveness:</i> transactions are eventually included $\frac{1}{2}$	Validity $\frac{1}{2}$

Blockchain -“Abstract” definition

Pro:

- + includes distributed systems aspects
- + formal reasoning

Con:

- does not capture full protocol
- proofs for strong system models

Blockchain Consensus

A closer look at how blockchain consensus achieves decentralized consensus

Blockchain Consensus

- Will describe Blockchain consensus by explaining how:
 - 1) Peers agree on a common prefix of blocks
 - 2) Proof-of-work is used in this context
 - 3) Double spending is avoided in Bitcoin

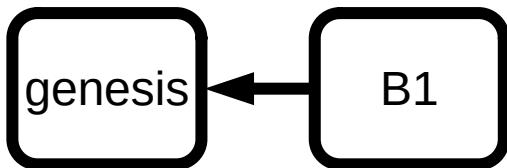
Blockchain Consensus

- the Bitcoin blockchain has a fixed starting point called *genesis block*

genesis

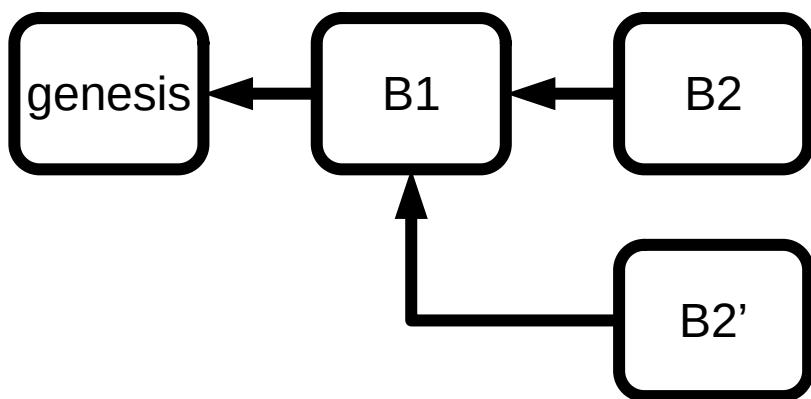
Blockchain Consensus

- newly mined blocks are appended
- the blockchain is an *append-only* storage
- Uses hash pointers to point to previous blocks



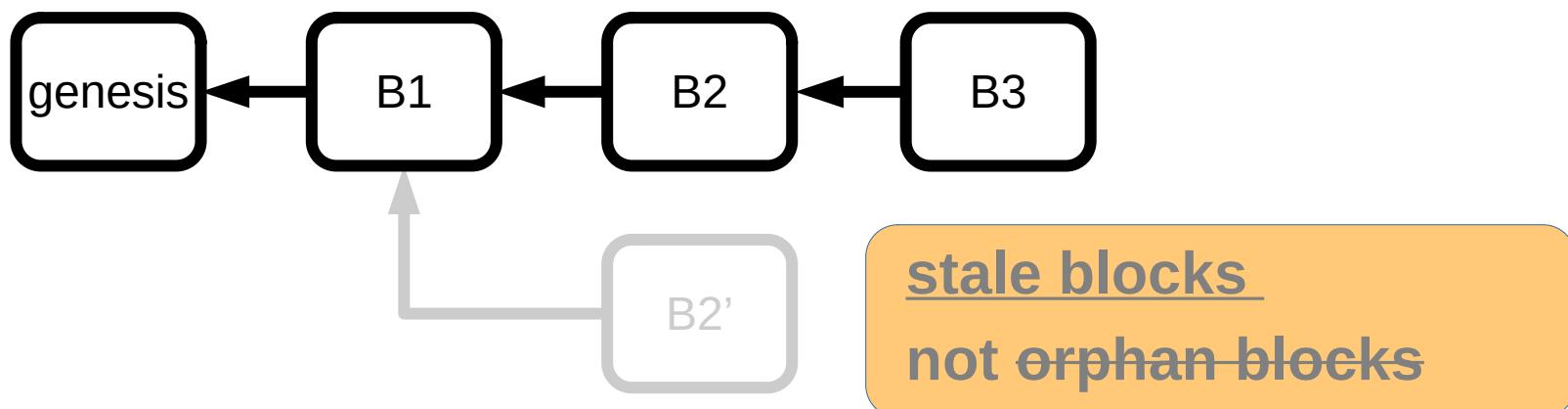
Blockchain Consensus

- it is possible that different valid blocks are mined at the same time
- this is called a *blockchain fork*



Blockchain Consensus

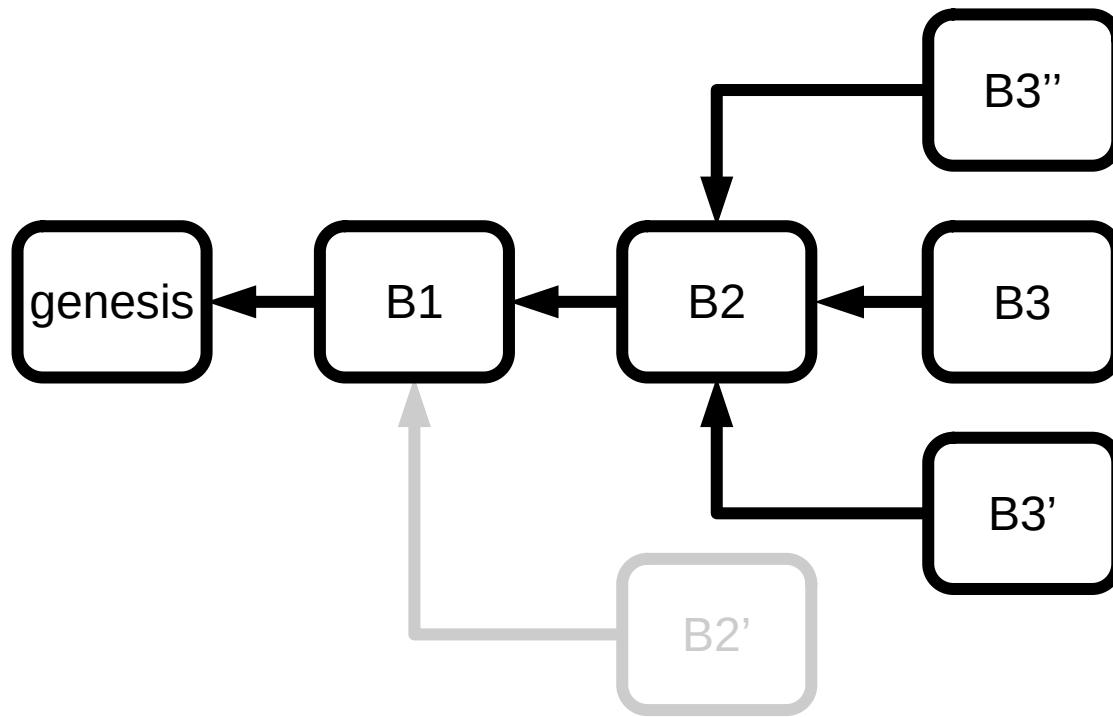
- Eventually one chain longer than the other
- The **longest chain*** is always the valid one (i.e. main chain)



[*] actually it is the most difficult chain that required the most hashing power to create

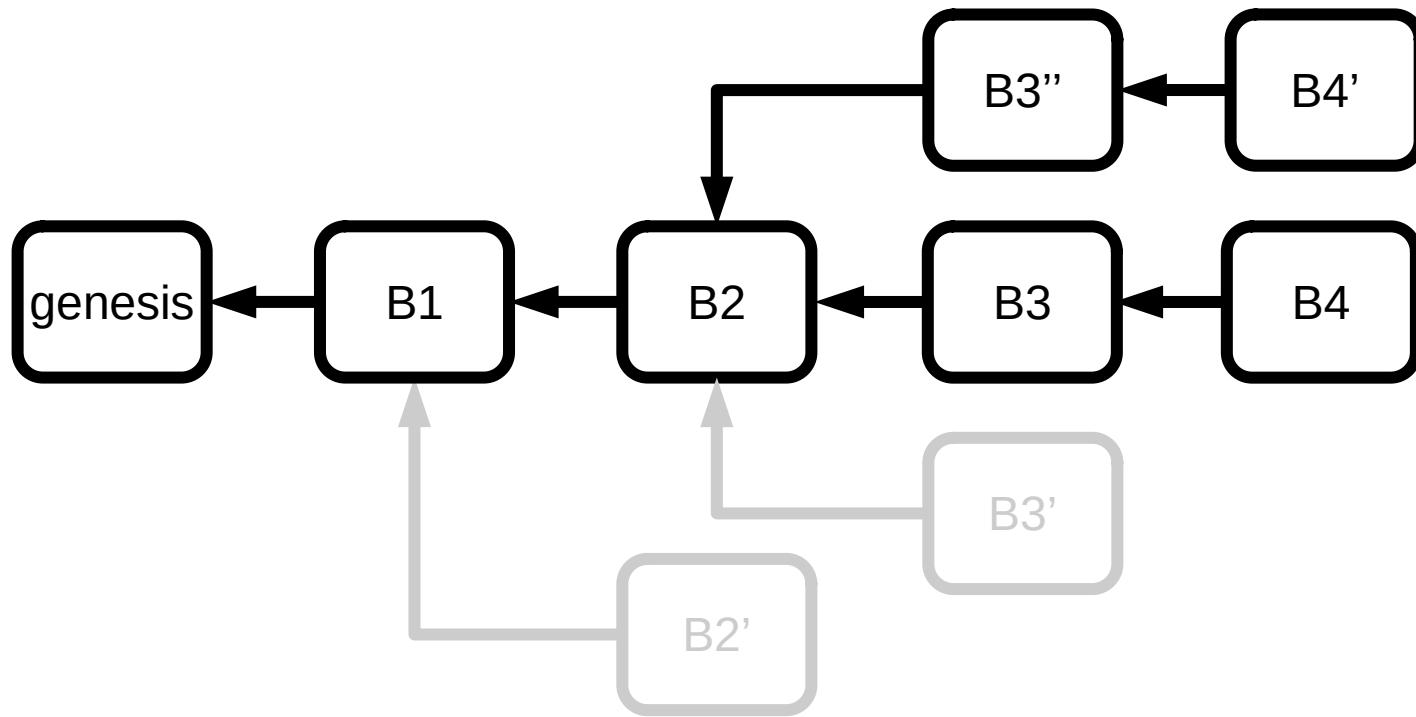
Blockchain Consensus

- there exists the possibility of multiple forks at the same time



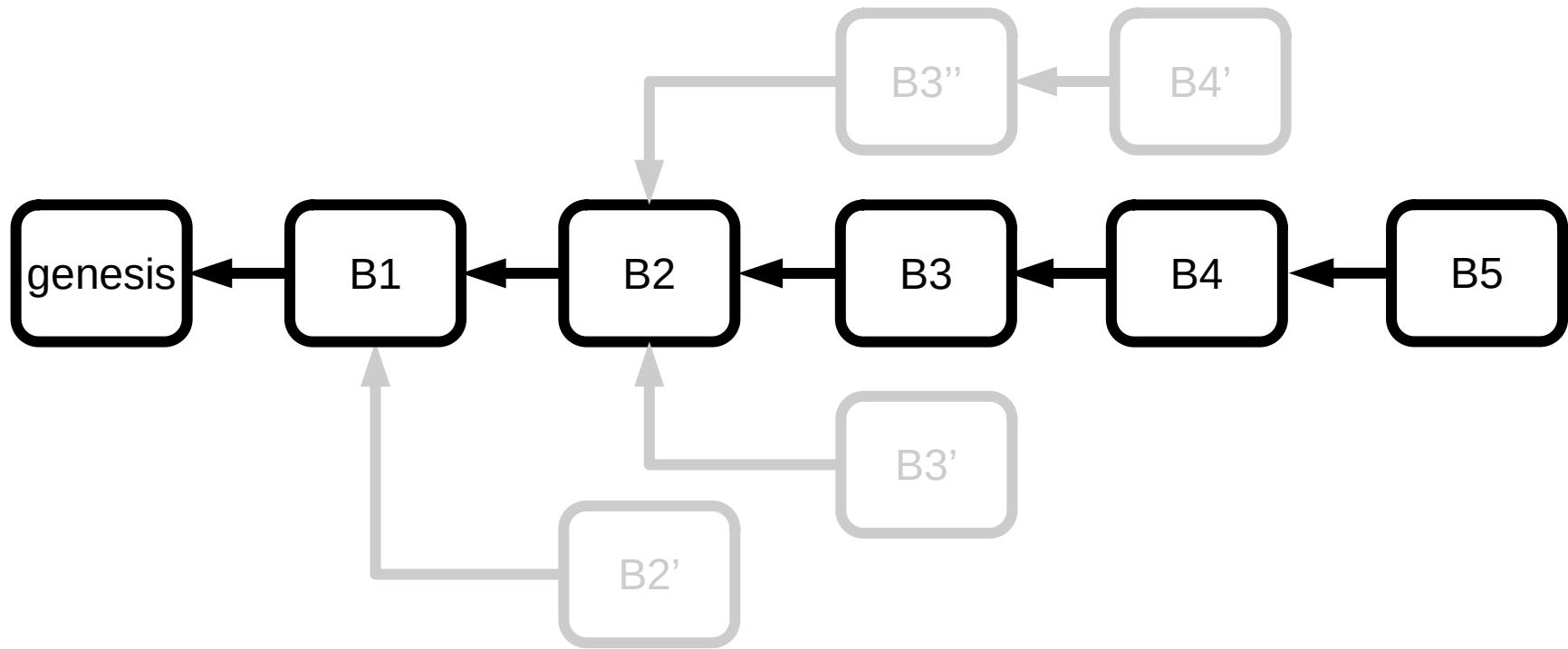
Blockchain Consensus

- the resolution mechanism is the same



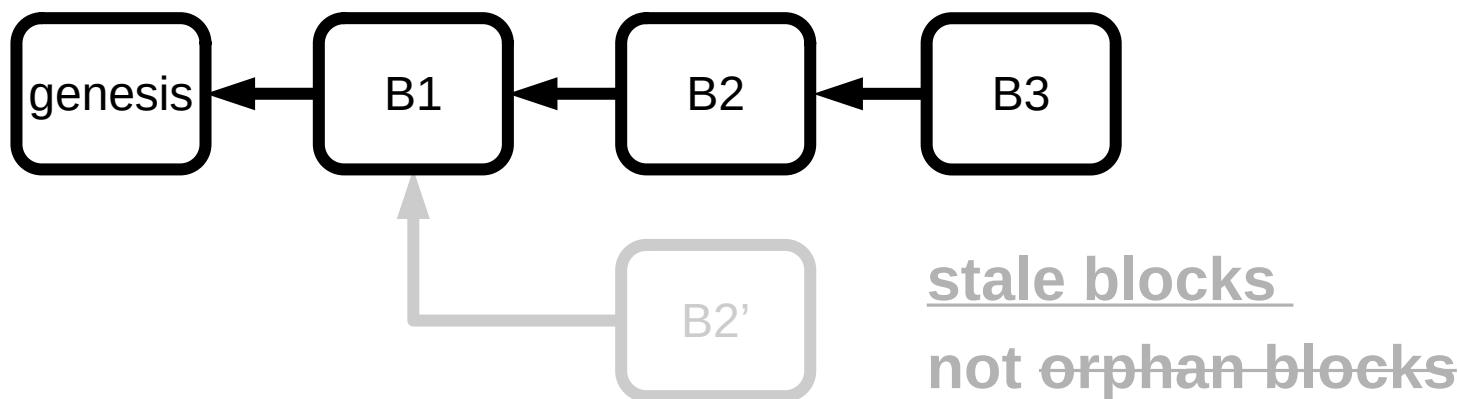
Blockchain Consensus

- the resolution mechanism is the same



Blockchain Consensus

- Eventually one chain is longer than the other
- The **longest chain*** is always the valid one
(i.e. main chain)

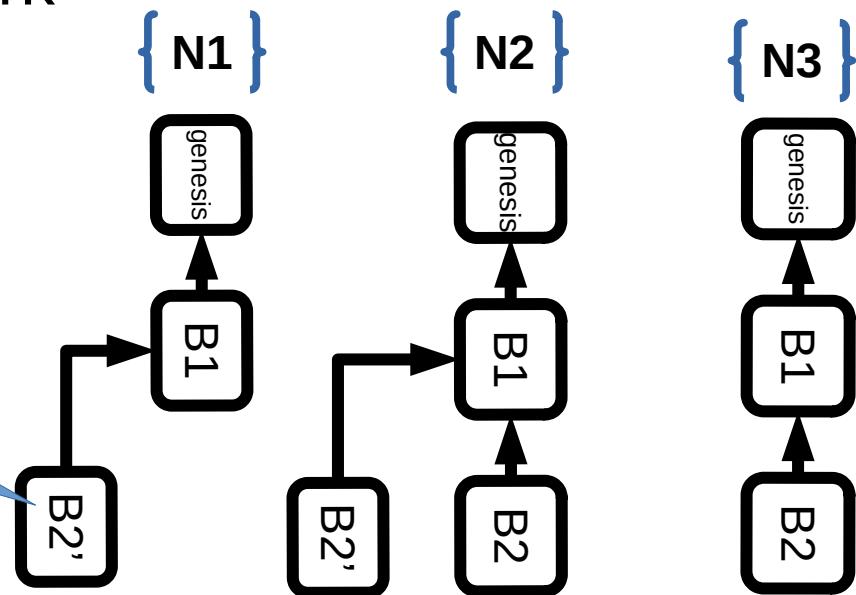


[*] actually it is the most difficult chain that required the most hashing power to create

Blockchain Consensus

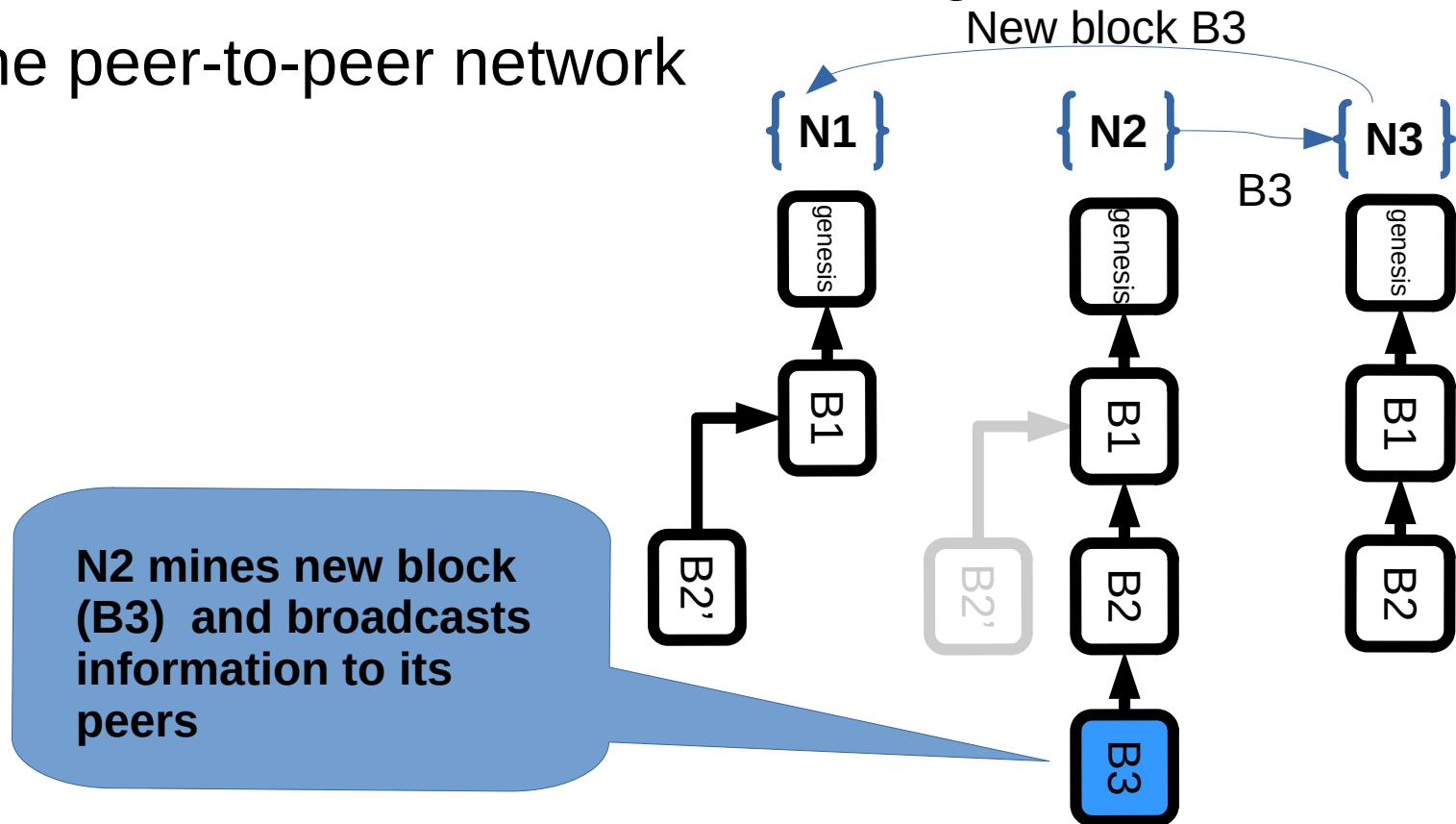
- Every full node has their own local copy of the blockchain
- Information is disseminated through the peer-to-peer network

Local copies of the Blockchain may not be exactly the same



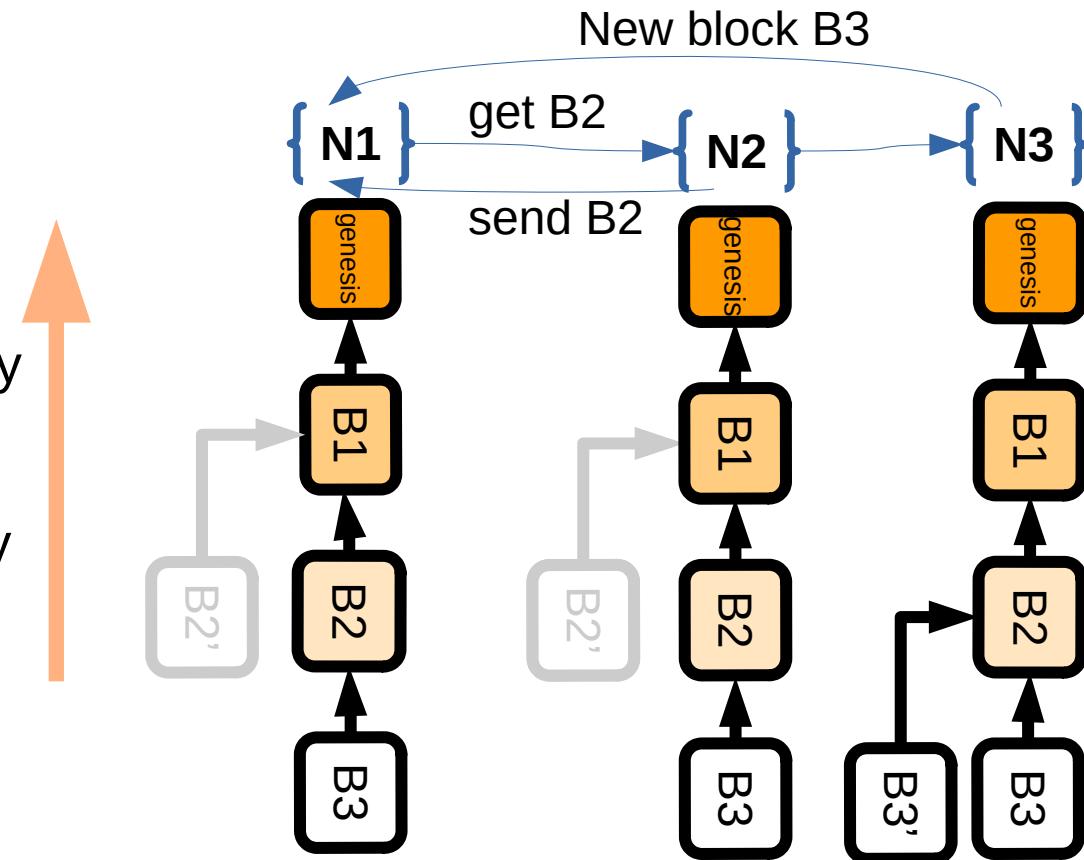
Blockchain Consensus

- Every full node has their own local copy of the blockchain
- Information is disseminated through the peer-to-peer network



Blockchain Consensus

- Each node considers only the longest* chain as valid
- As the chain grows nodes eventually agree on a **common prefix of blocks**



In Bitcoin the probability that nodes agree on a common prefix increases exponentially with the number of confirmation blocks

Blockchain Consensus

- Simply following the longest chain without limiting the generation of Blocks is a problem

Blockchain Consensus

- Simply following the longest chain without limiting the generation of blocks is a problem

Question:

Why?

Blockchain Consensus

- Simply following the longest chain without limiting the generation of blocks is a problem
 - Generating longer chains is relatively easy
 - Nodes can spam the network
 - Can pretend to be many nodes - “Sybil attack”

→ Proof-of-work solves these problems

PoW in Blockchain Consensus

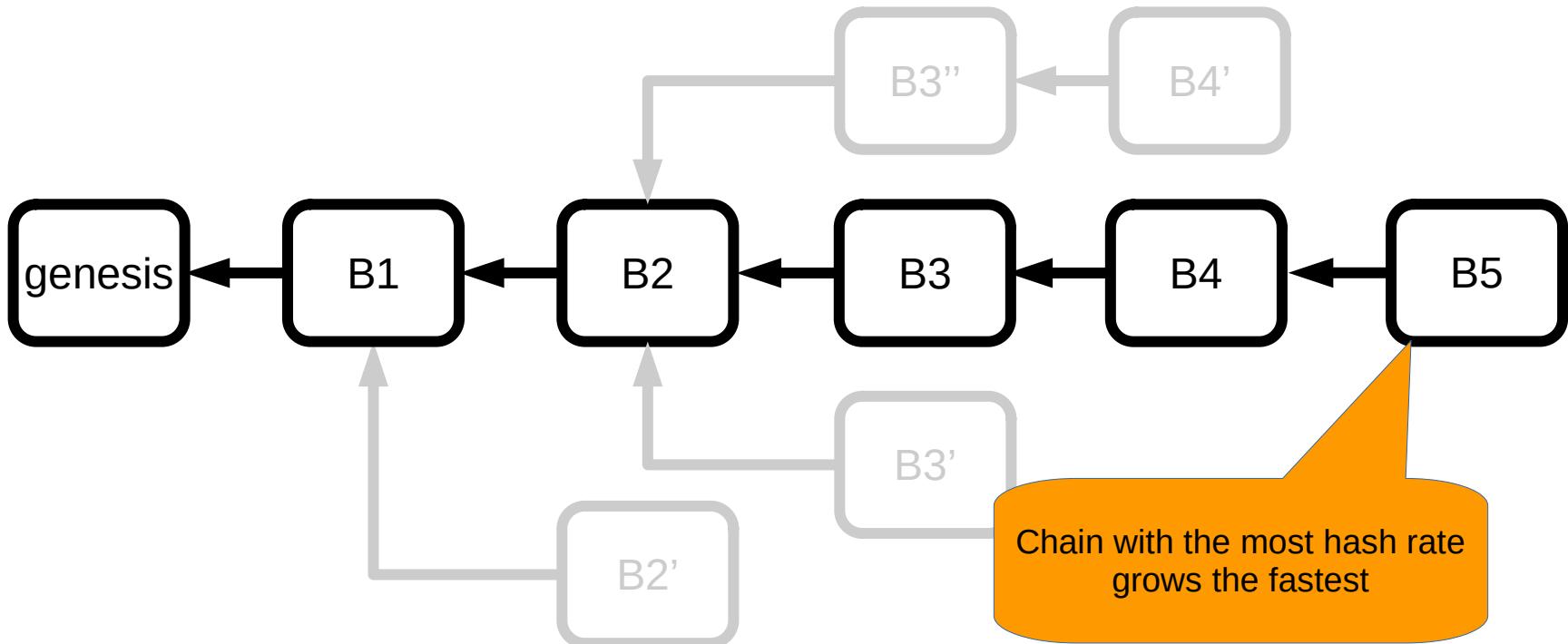
- Votes are cast with computing power
- An attacker can't easily fake PoW
- Honest nodes will append new blocks to the longest chain (that they know of)
- Eventually nodes will agree on a common prefix of the chain*
- Miners are actually rewarded for their participation (incentive to act rationally)
- Miners can join and leave the system

*If there is an honest majority of hash power

PoW in Blockchain Consensus

Assumption:

- If >50% of nodes are honest and append to the longest chain it will eventually grow faster than any competing fork
- Mining is probabilistic so accidental forks should eventually resolve



PoW in Blockchain Consensus

- Mining is principally anonymous
 - Only have to broadcast valid block once found
 - Would require anonymity network for nodes (TOR etc.)
- Mining nodes can join and leave the system without having to announce it
- Assumes that participants are not eclipsed in the network
 - Else they do not see updates and hence can not follow the actual longest chain

This enables dynamic membership

Blockchain Consensus

Early works on the characteristics of Blockchain consensus [1] showed that under the assumption of a *Synchronized network*:

- Theoretical fault tolerance of PoW Blockchains:
 - Attacker < 50 % hash rate

This **does not** take into account new types of attack e.g. block-withholding attacks

Attacker hash rate recommendation (for Bitcoin):

- < 25% to prevent selfish mining
- < 38.2 % to prevent 1 block forks

Double spending in Bitcoin

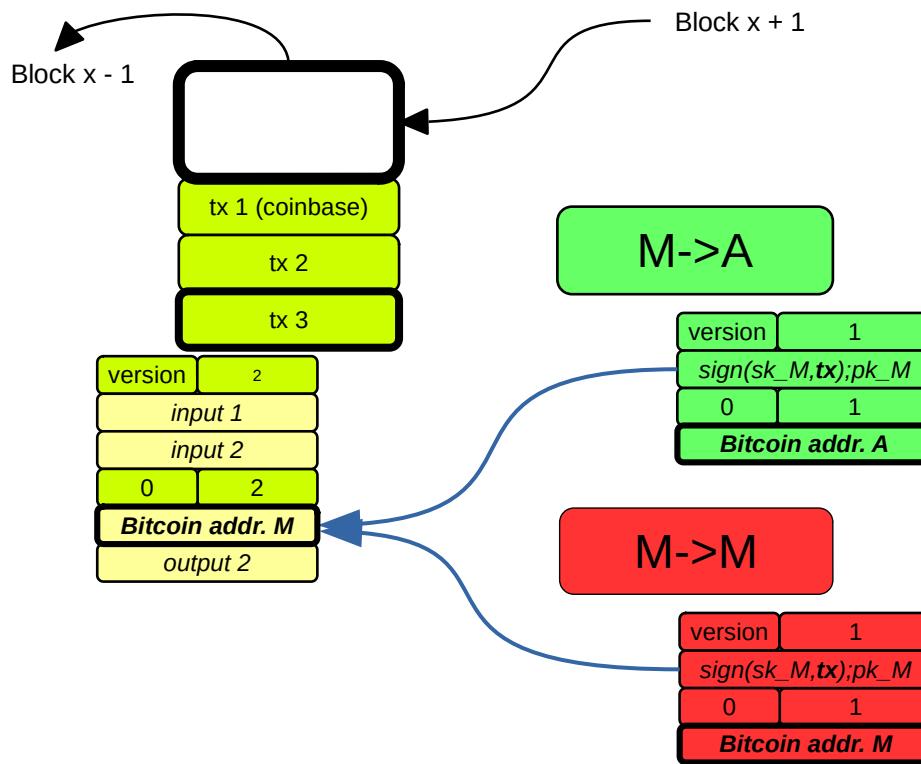
- To protect from double spending the Bitcoin client needs to enforce the following rules:
 - Only **previously unspent** tx outputs may be used as inputs for follow-up transactions
 - **UTXO** (Unspent transaction output)
 - STXO (Spent transaction output)
 - The **order** of transactions is determined by their order in the blockchain

Double spending in Bitcoin

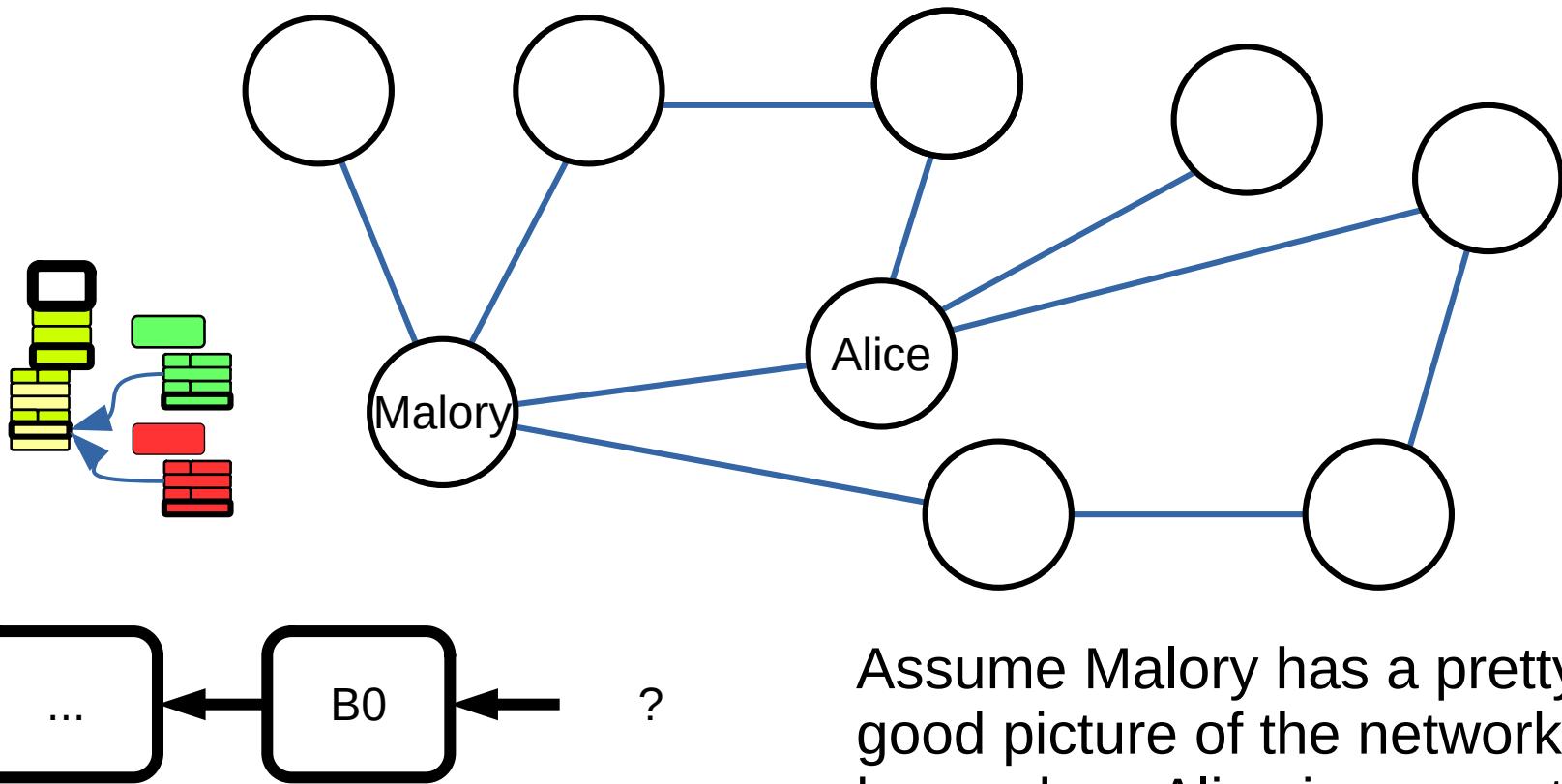
- an attacker (Malory) tries to trick a merchant (Alice) that she has actually payed
- Malory creates a tx and sends it to Alice M->A
- Malory also creates a conflicting tx that spends the same UTXO M->M
- The attack is successful, if Malory is able to convince the rest of the network that the coins are sent to an address which belongs to Malory

Double spending in Bitcoin

- Create conflicting tx which reference same UTXO



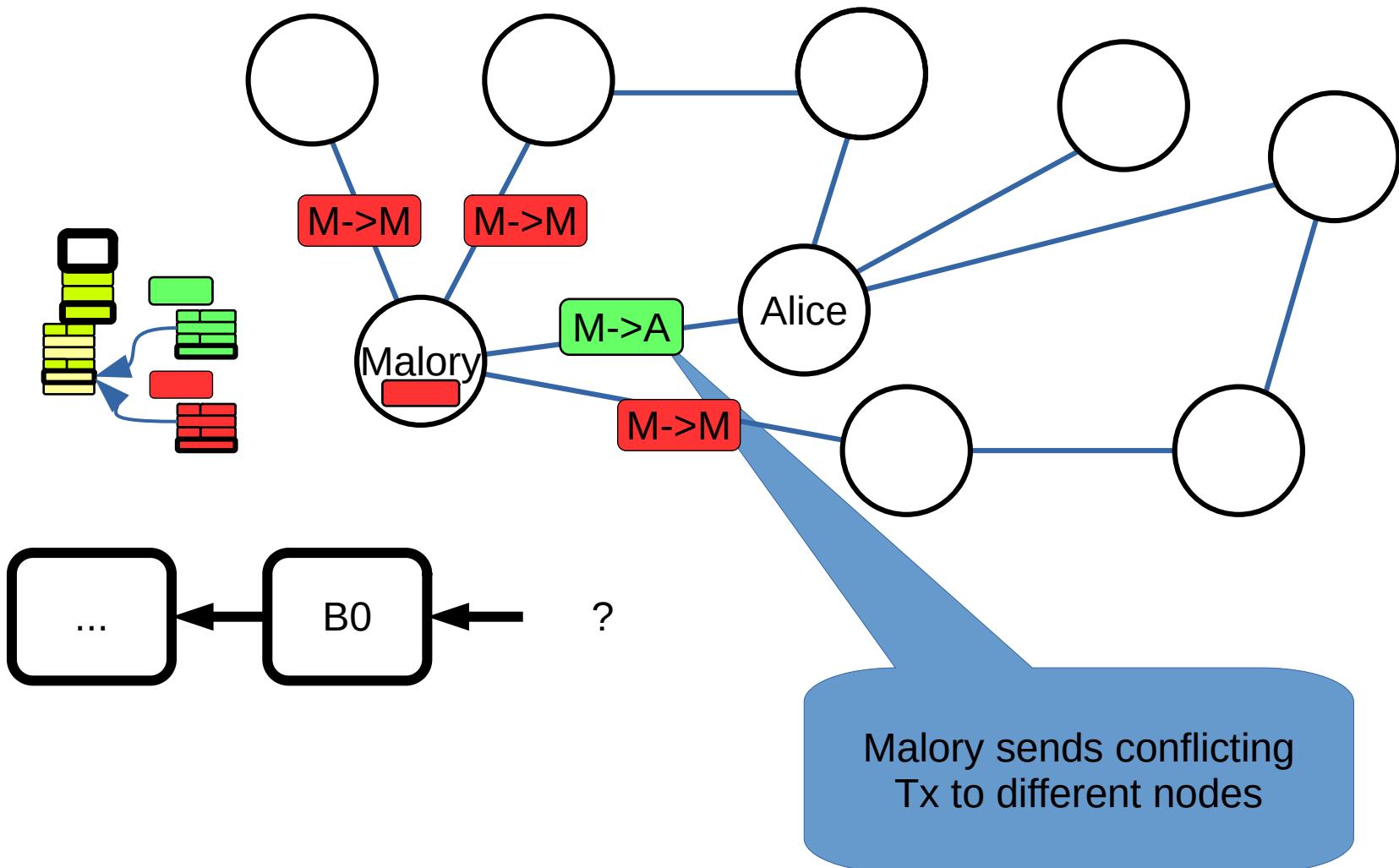
Double spending in Bitcoin



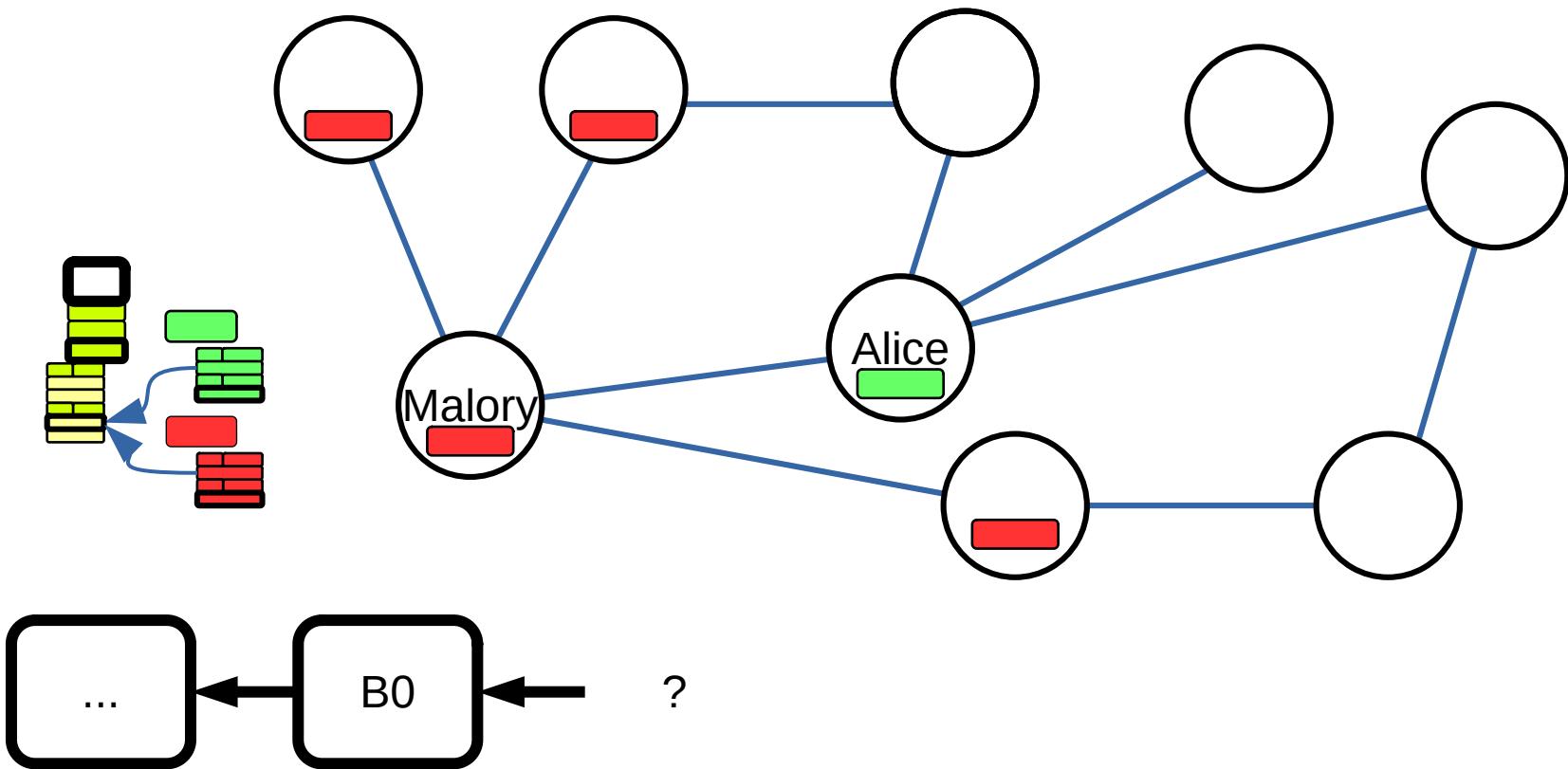
Assume Malory has a pretty good picture of the network and knows how Alice is connected.

Not necessarily required for double spending but helpful*

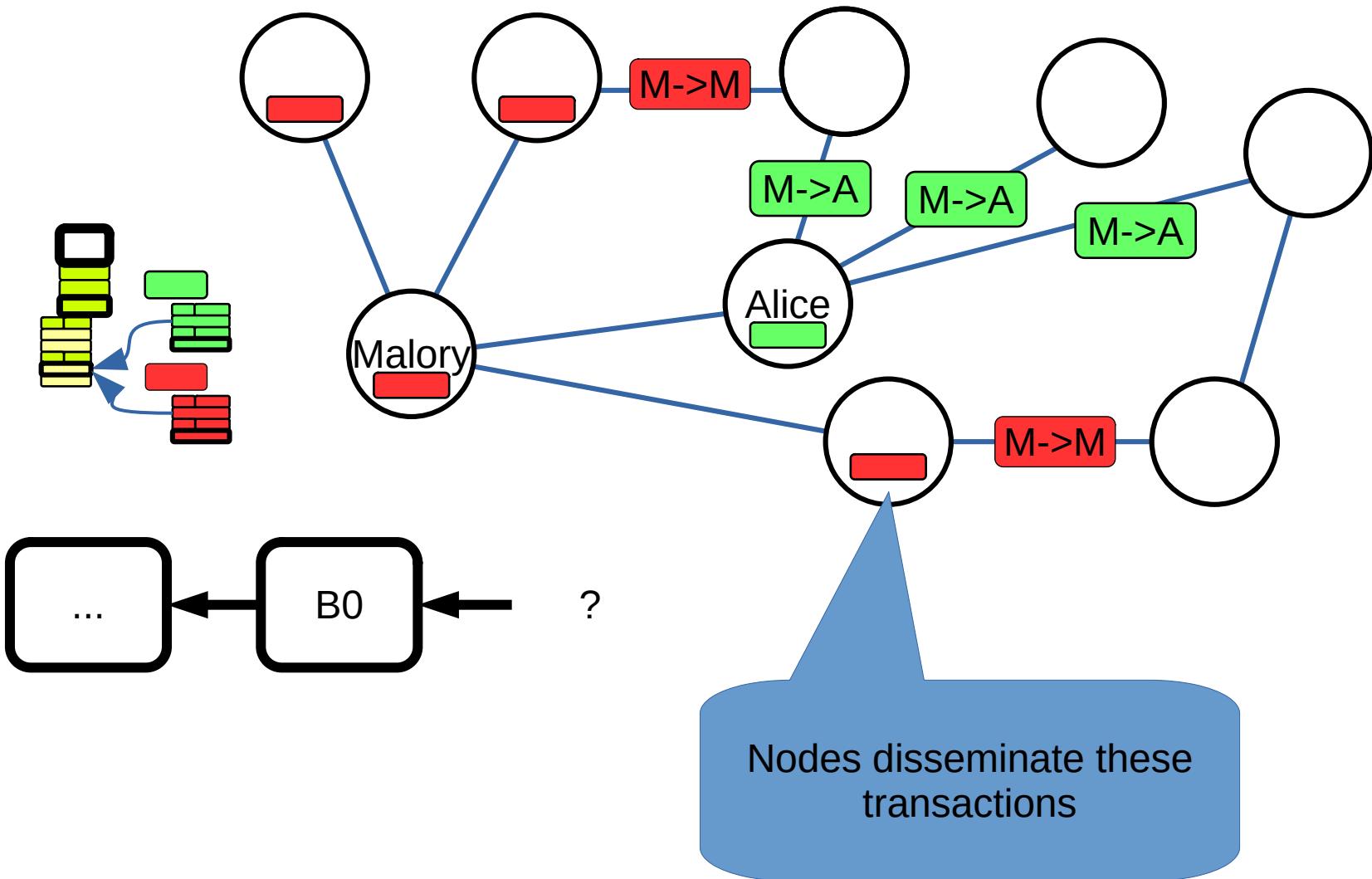
Double spending in Bitcoin



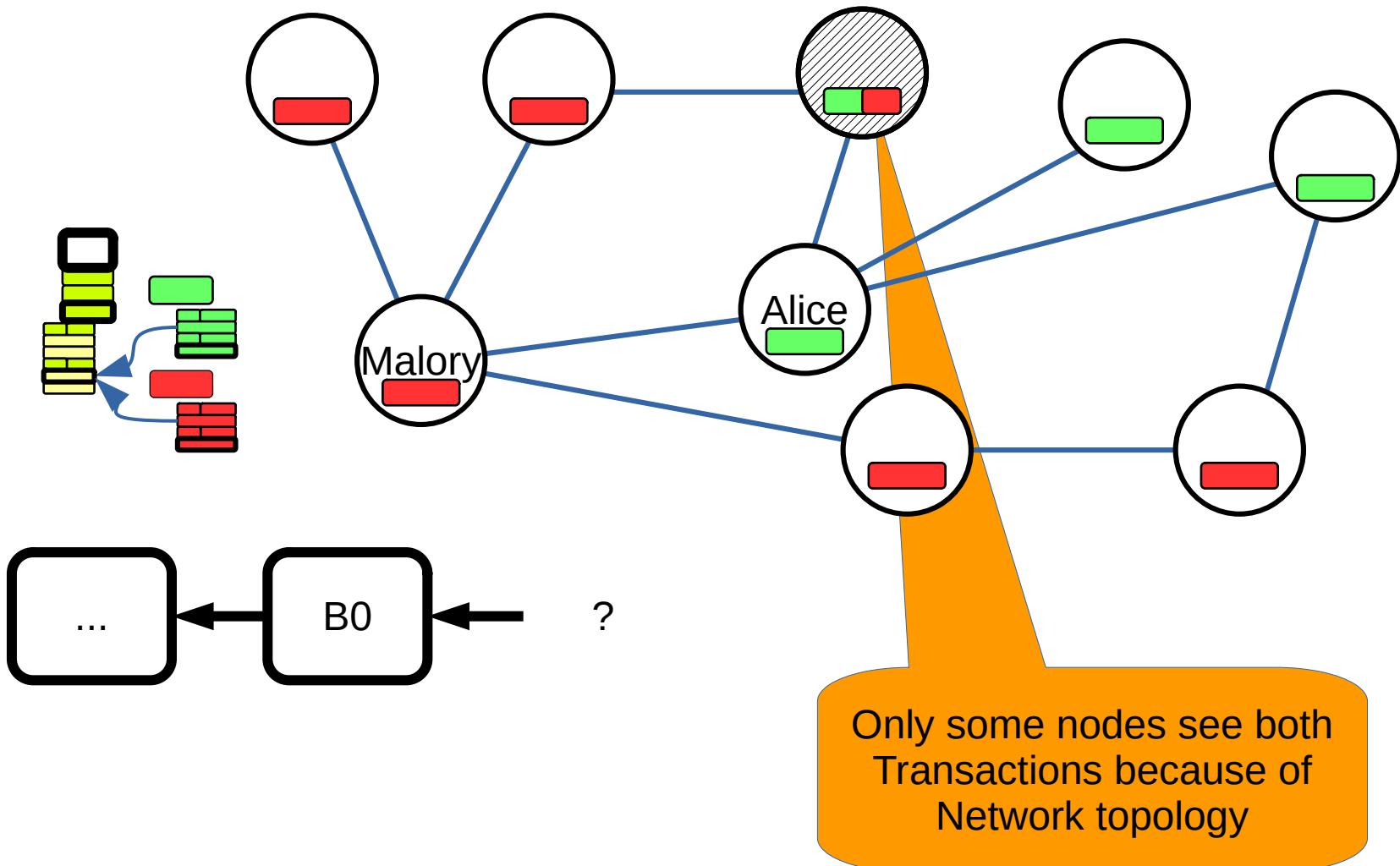
Double spending in Bitcoin



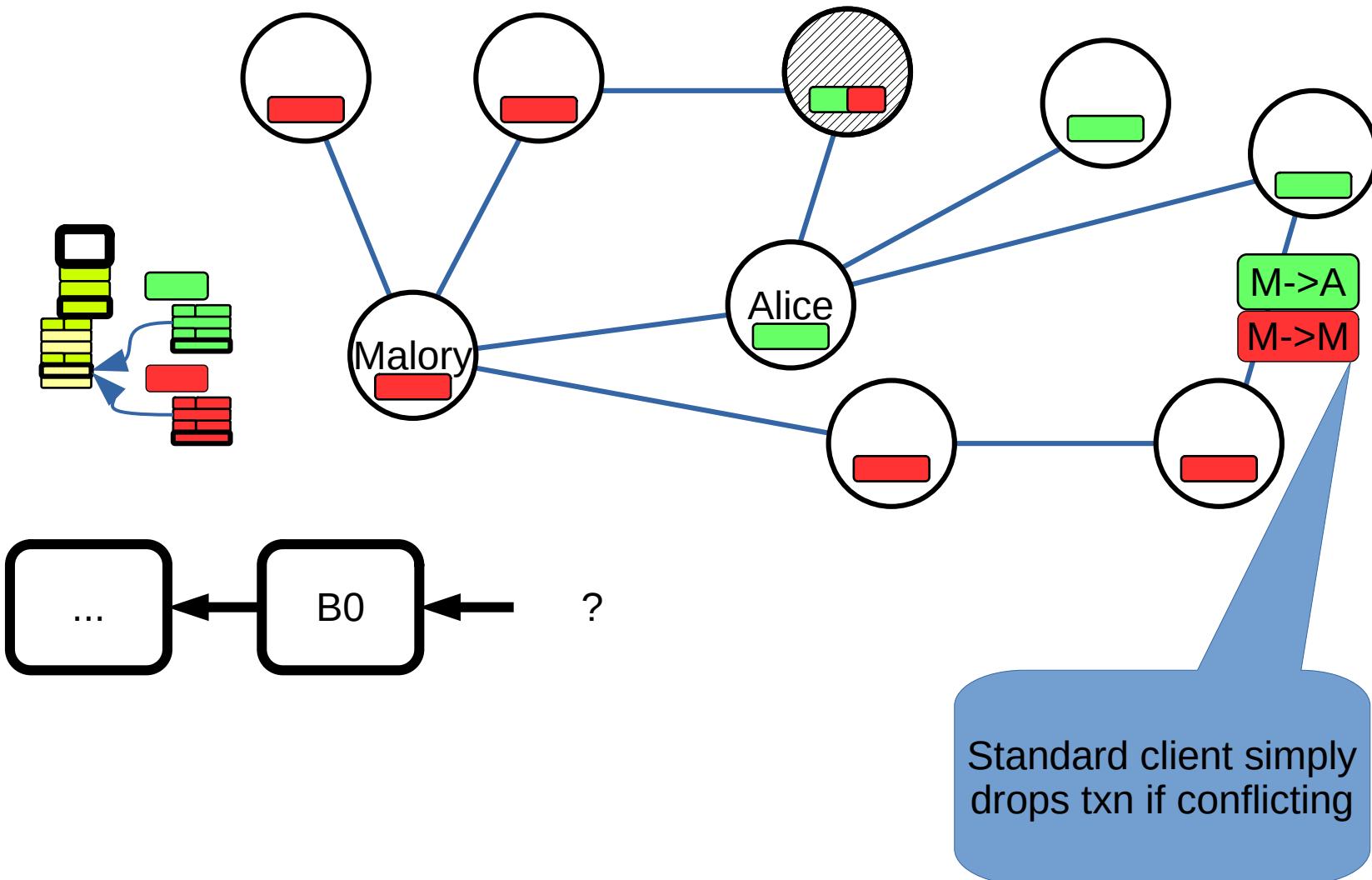
Double spending in Bitcoin



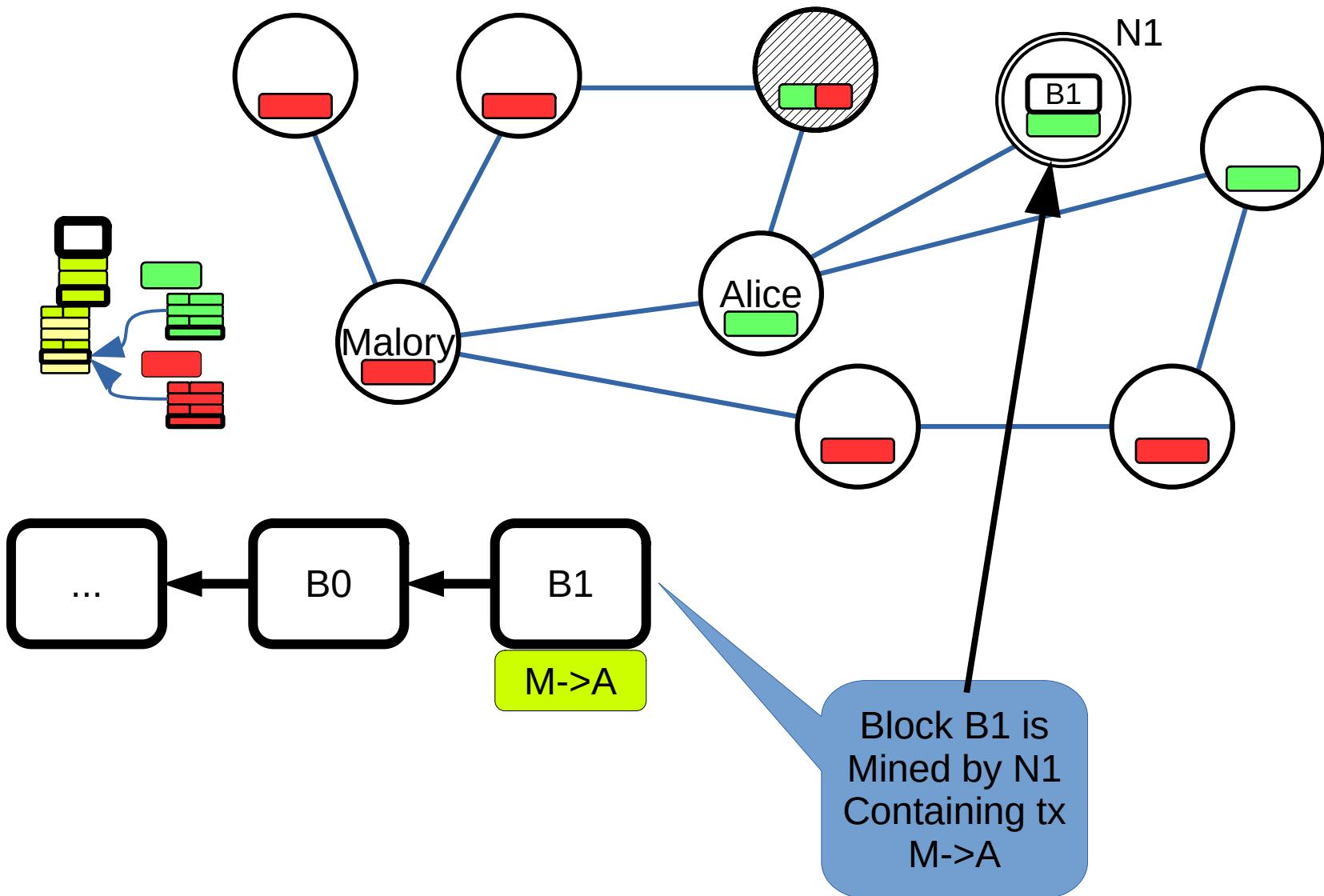
Double spending in Bitcoin



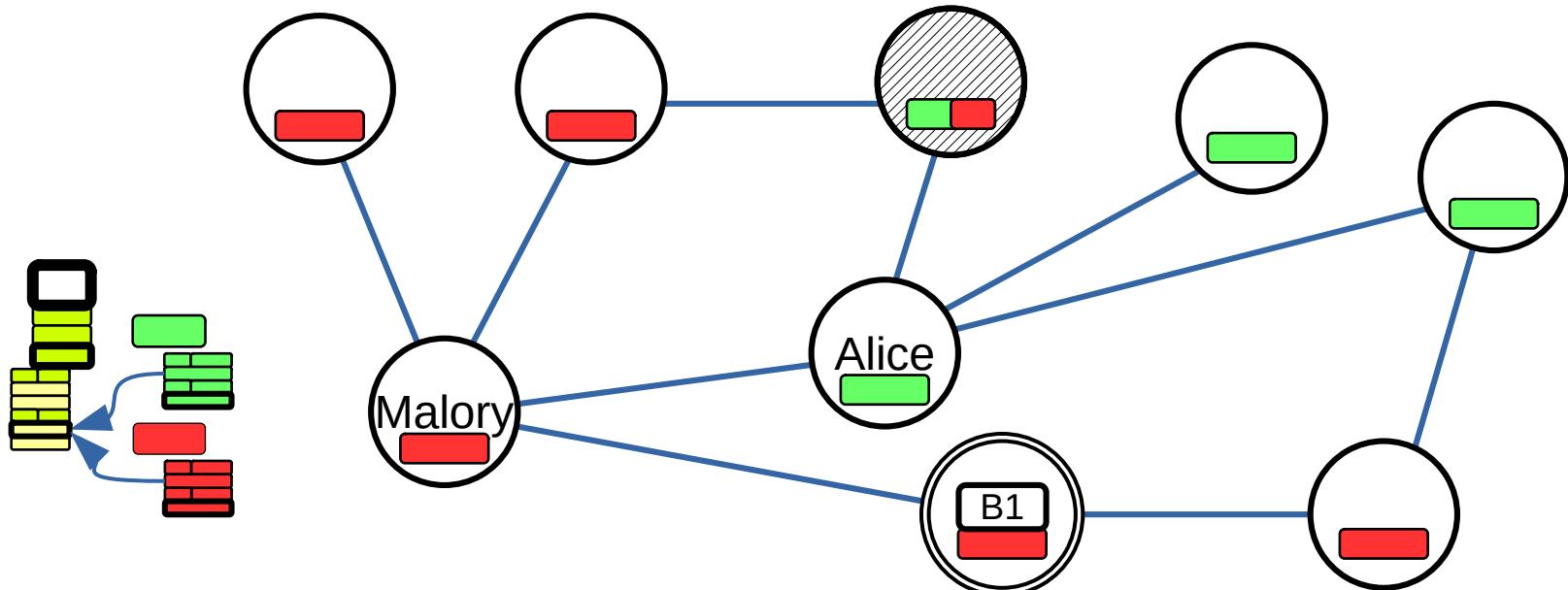
Double spending in Bitcoin



Double spending in Bitcoin



Double spending in Bitcoin

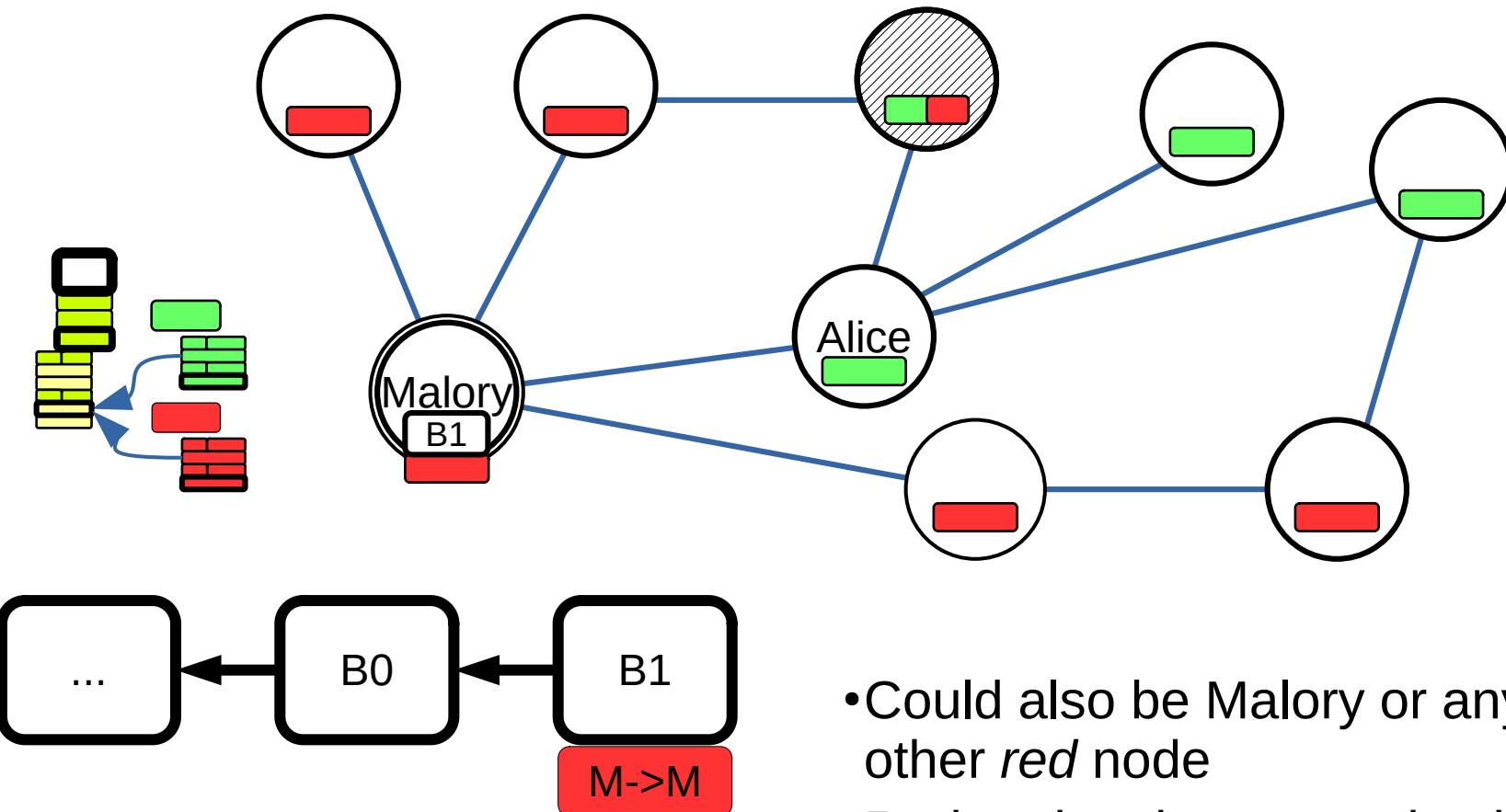


Alternatively:

Conflicting transaction is mined in B1

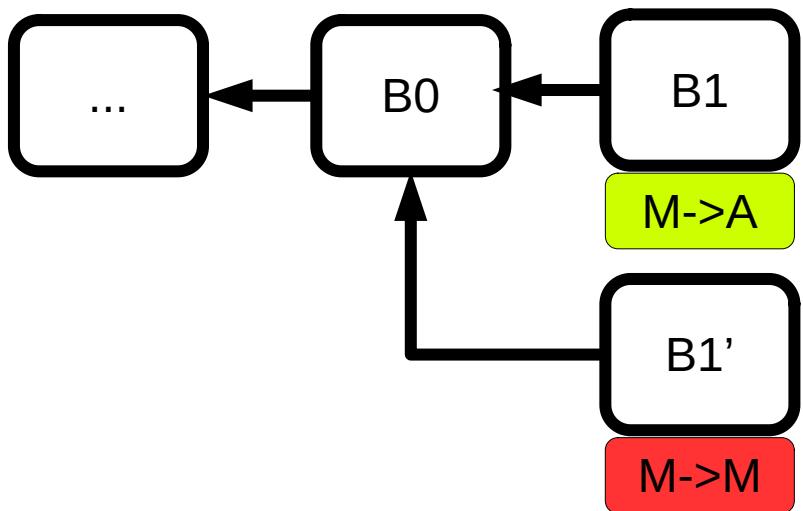
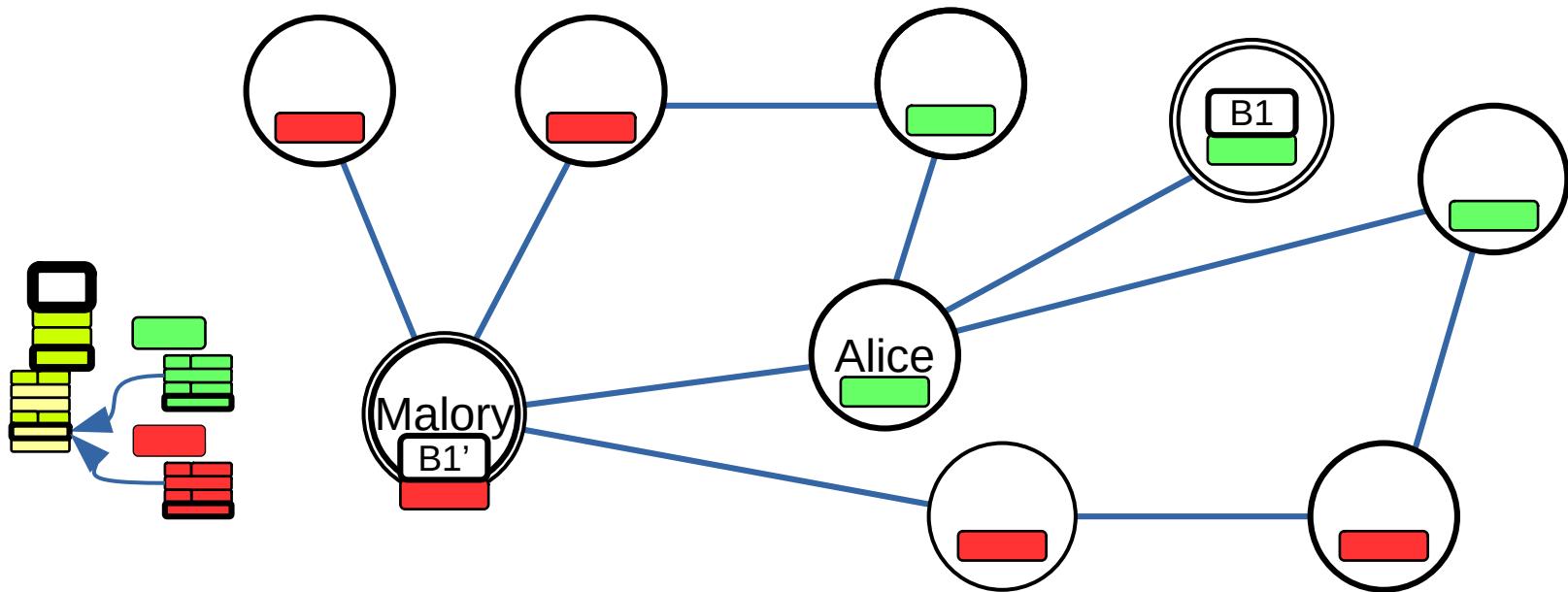
- If the merchant has already accepted the payment the **attack was successful**
 - (This is why 0-conf payments are dangerous)

Double spending in Bitcoin



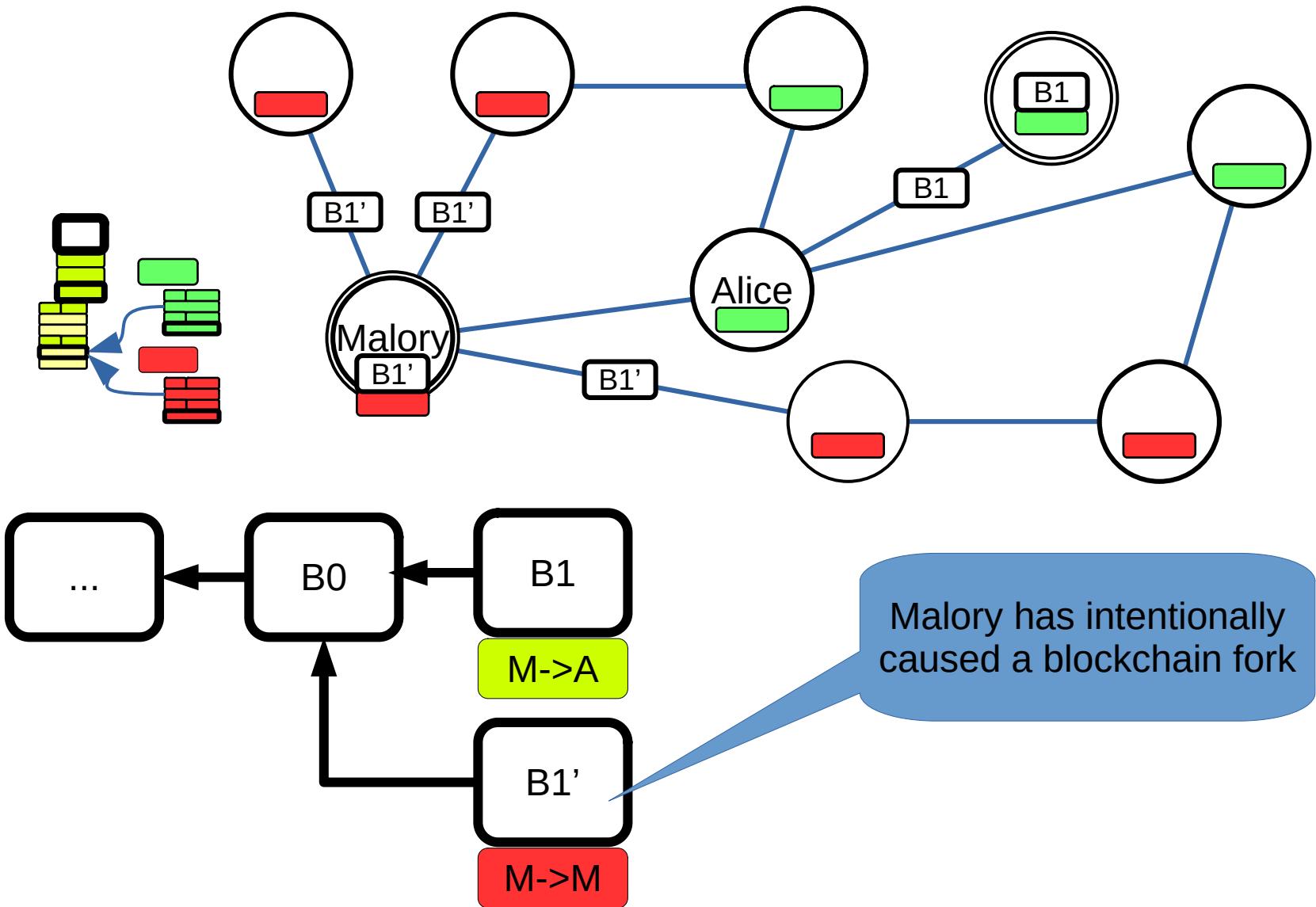
- Could also be Malory or any other *red* node
- Red nodes do not need to know that Mallory is attacking Alice

Double spending in Bitcoin

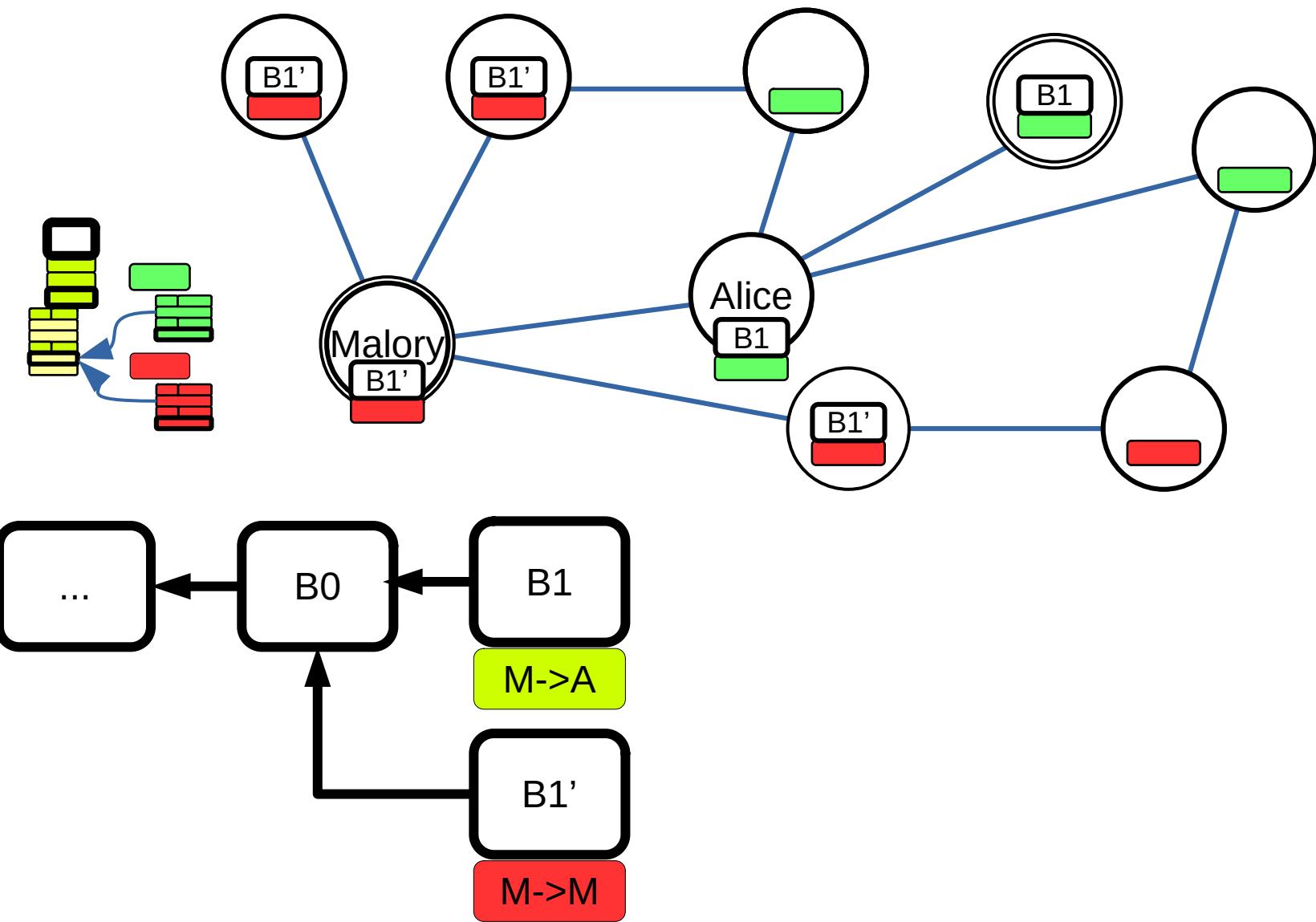


Assume Malory wont give up so easy and also mines a block

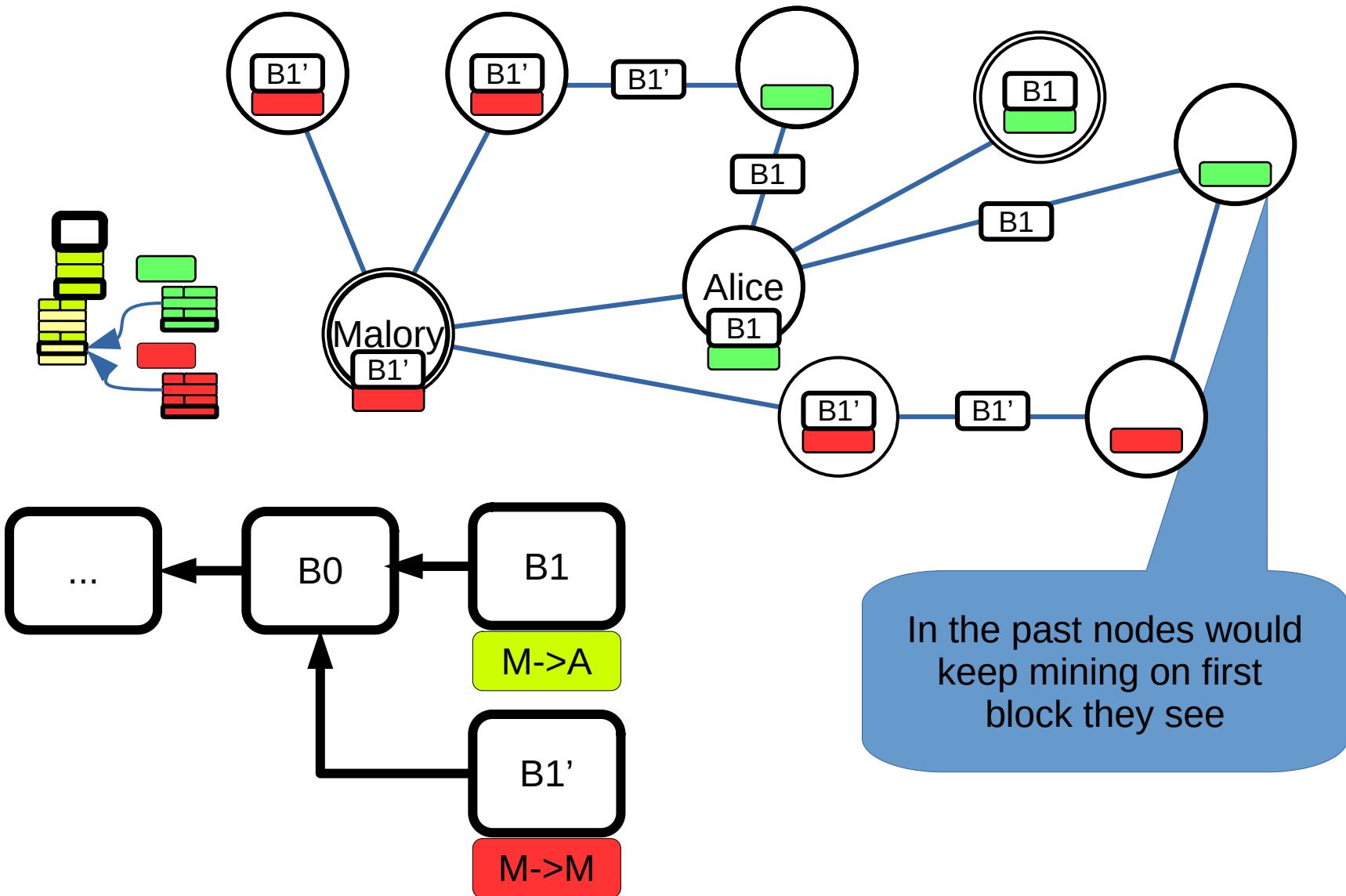
Double spending in Bitcoin



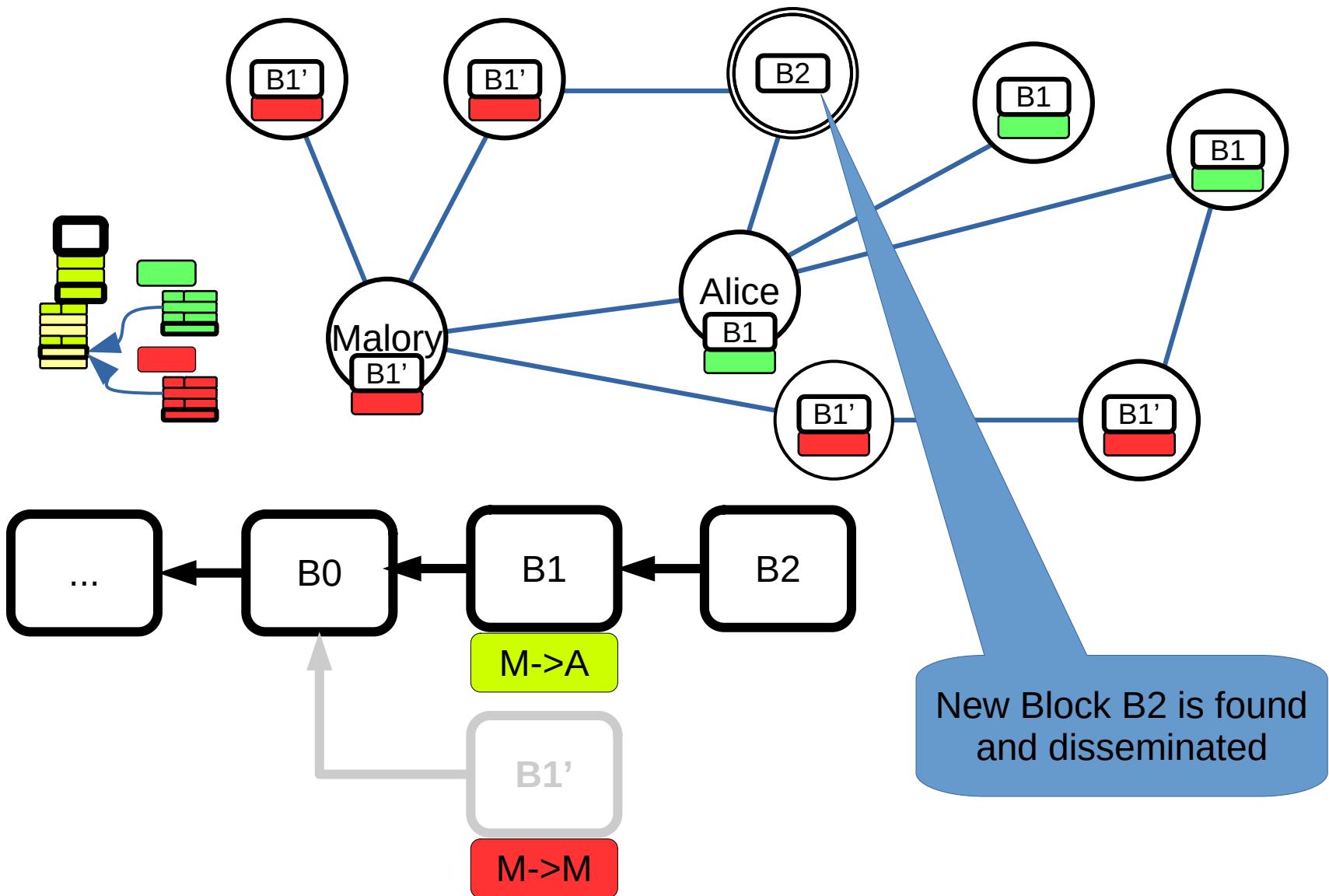
Double spending in Bitcoin



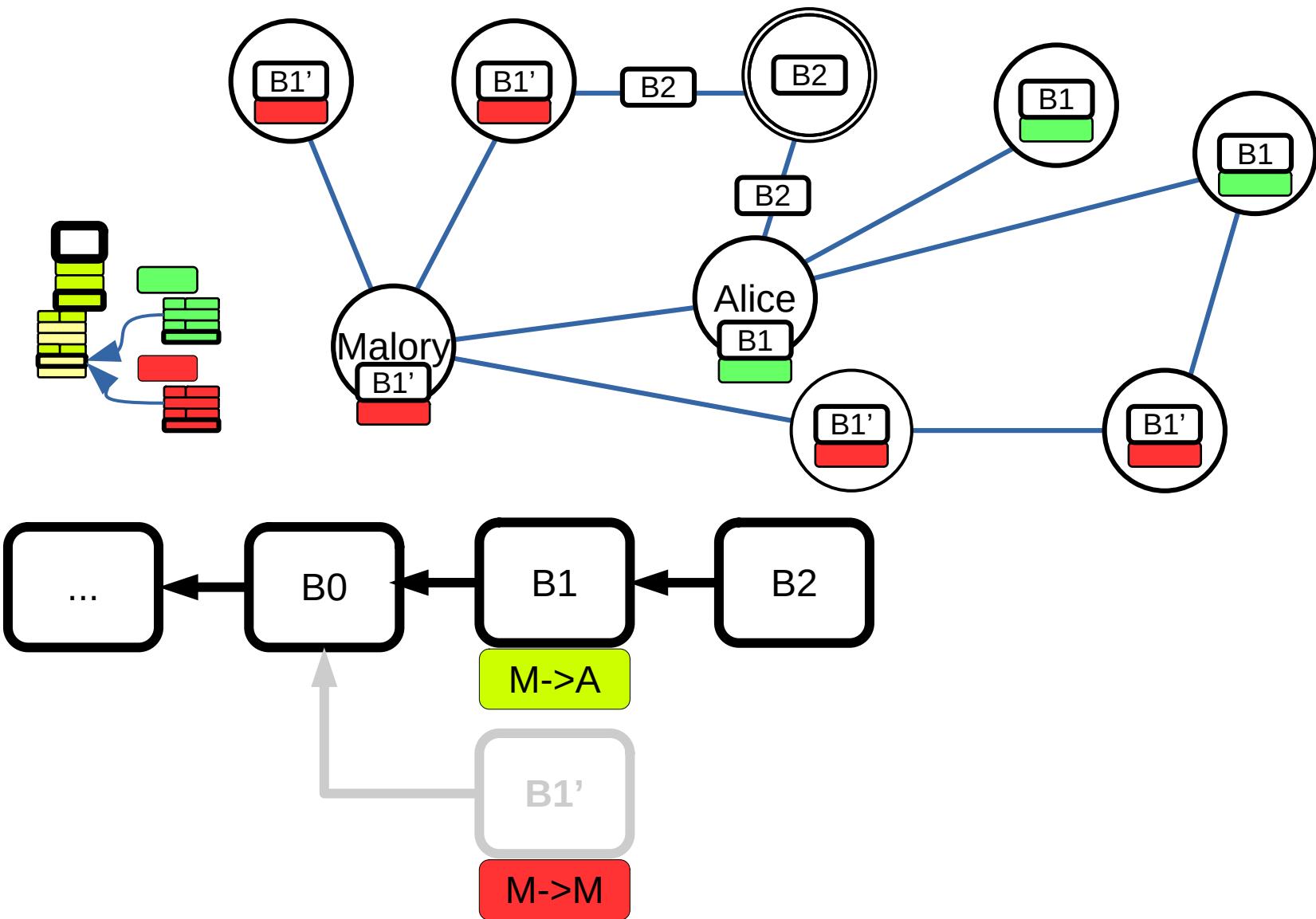
Double spending in Bitcoin



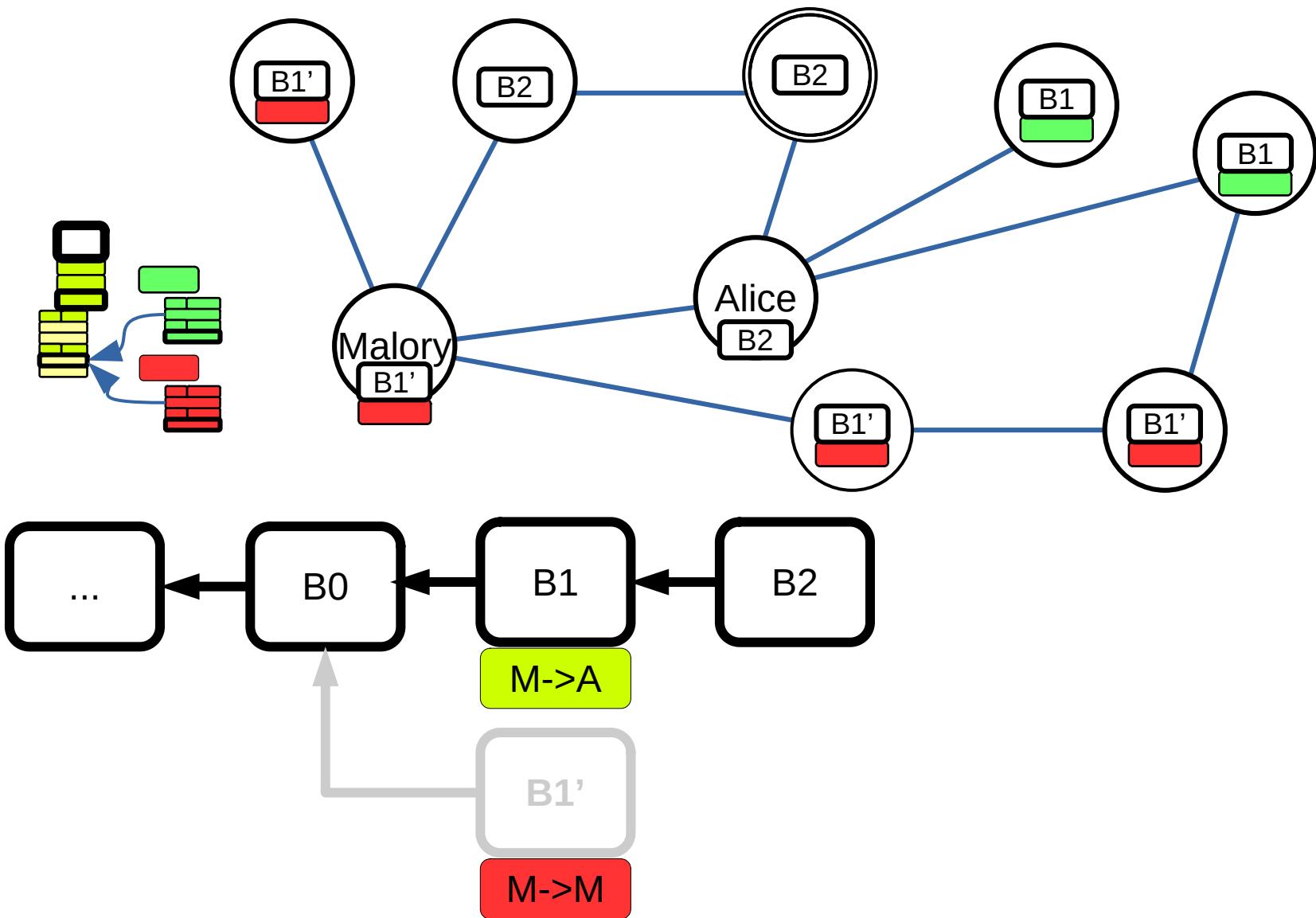
Double spending in Bitcoin



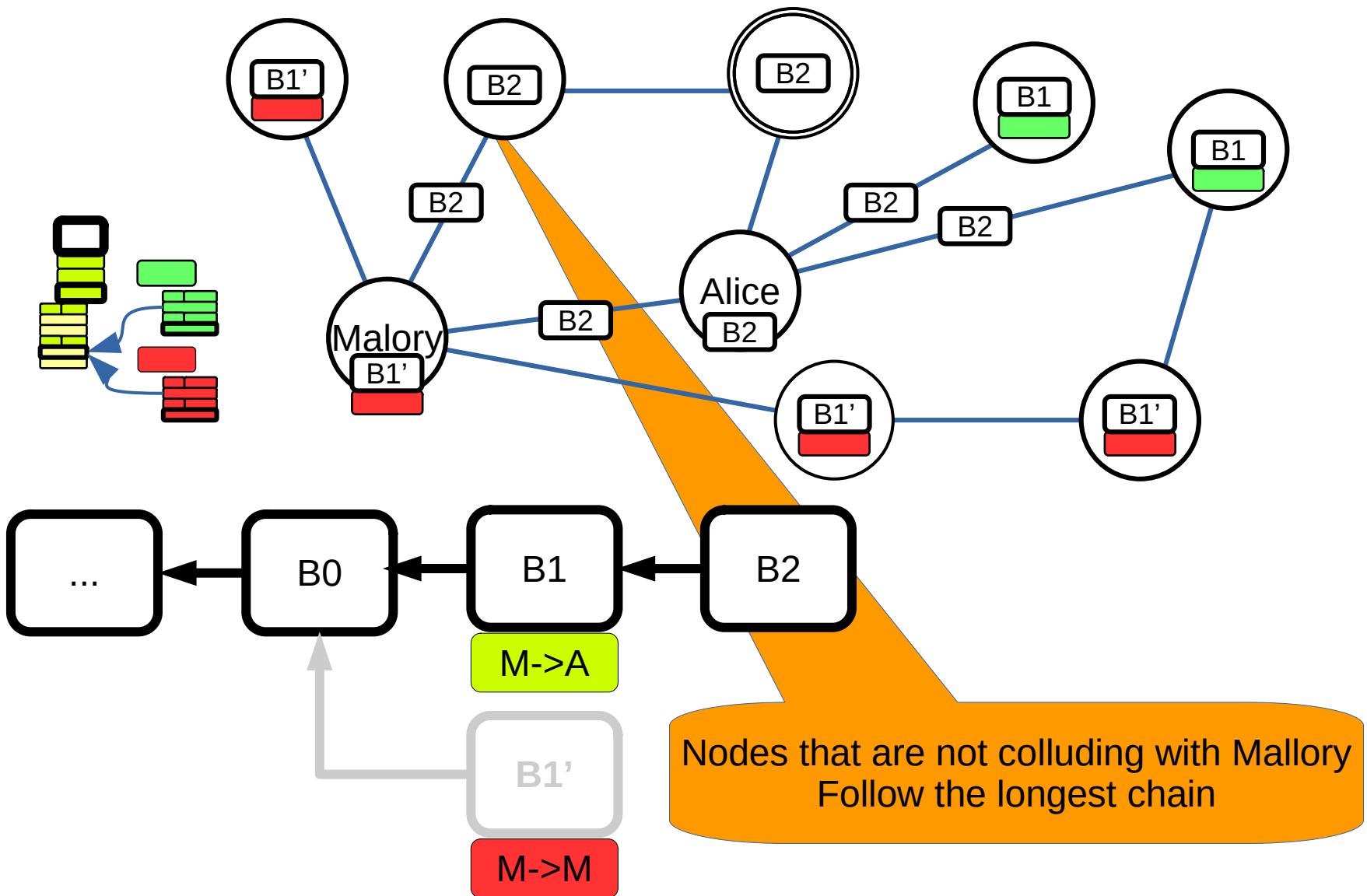
Double spending in Bitcoin



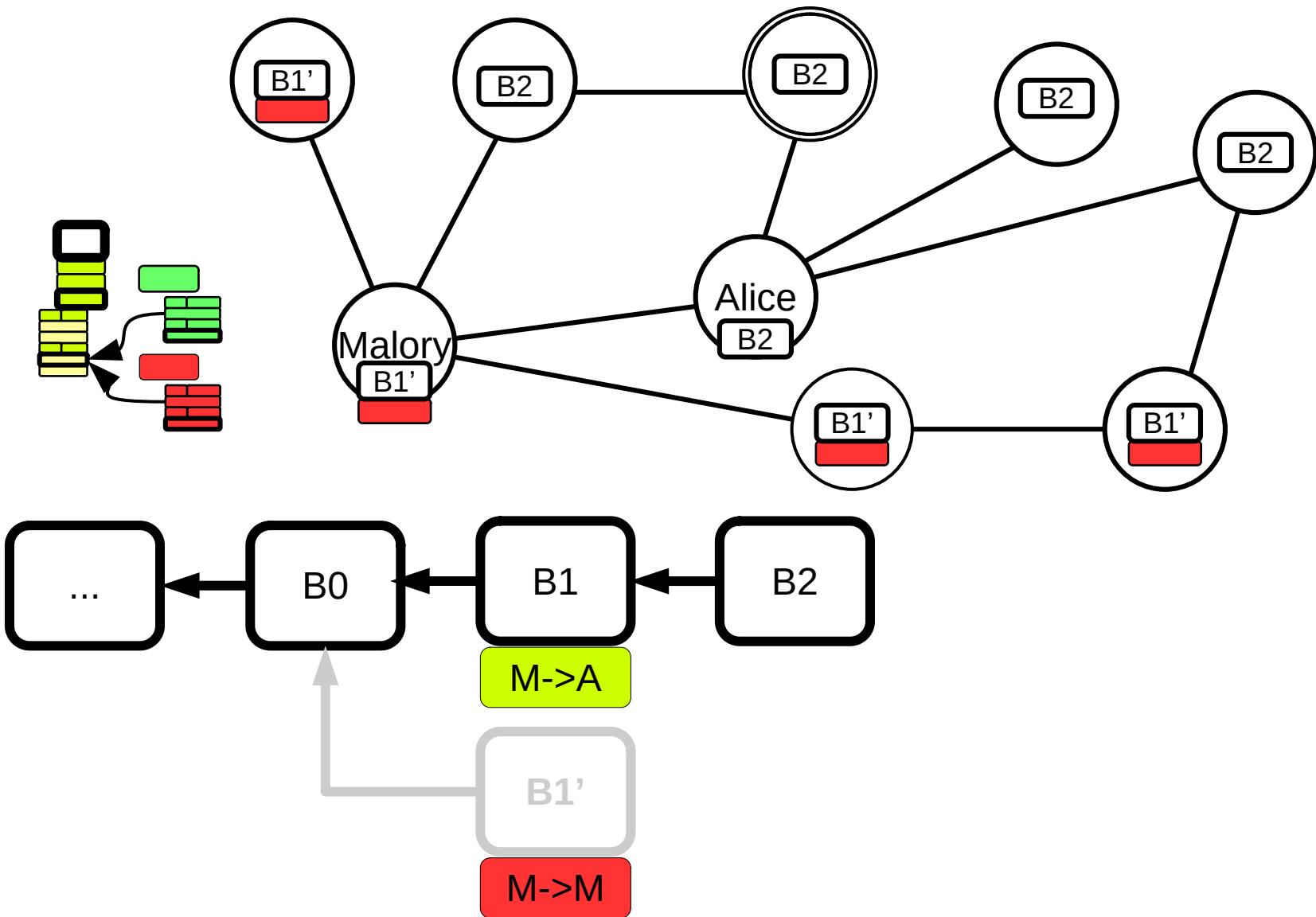
Double spending in Bitcoin



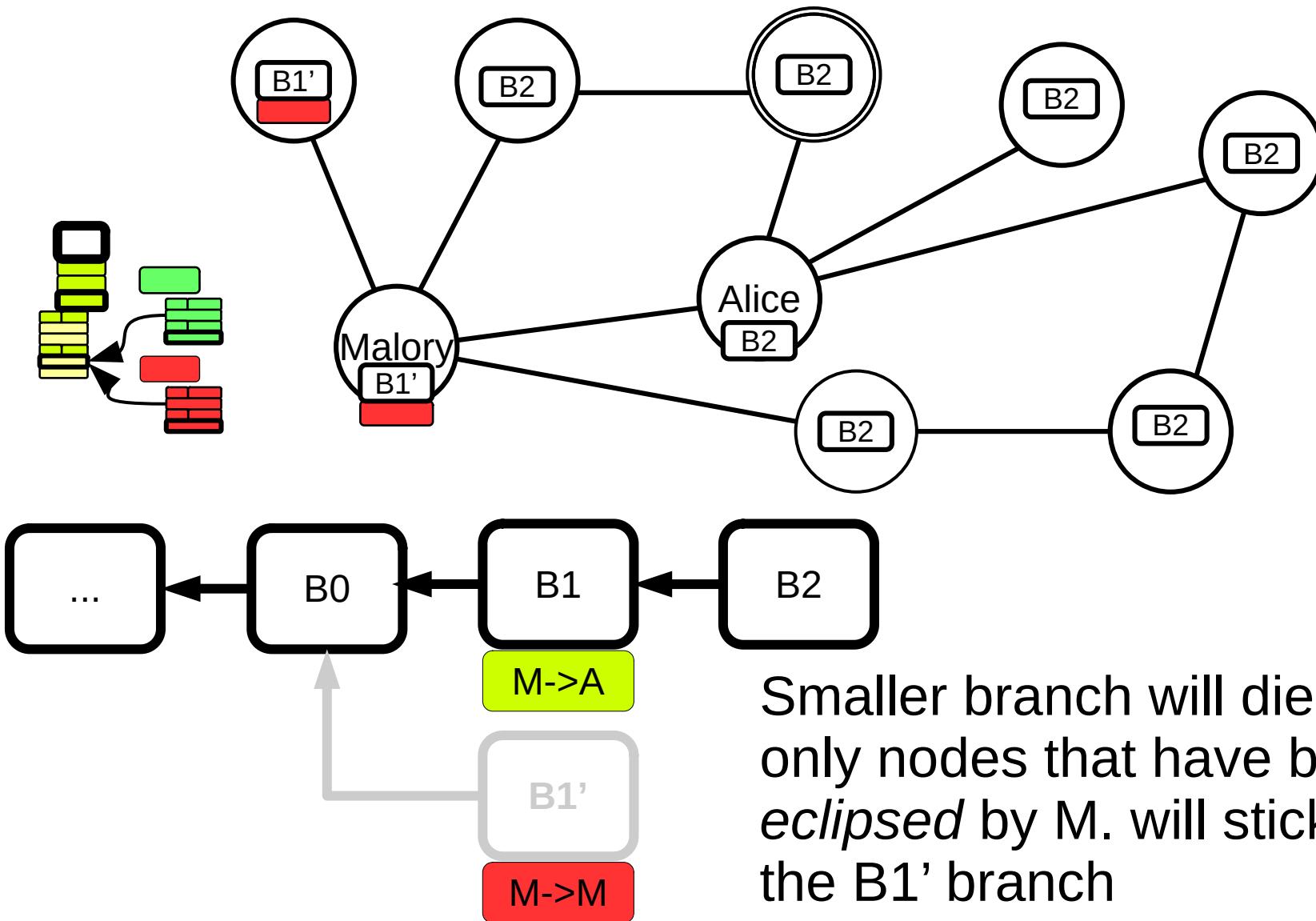
Double spending in Bitcoin



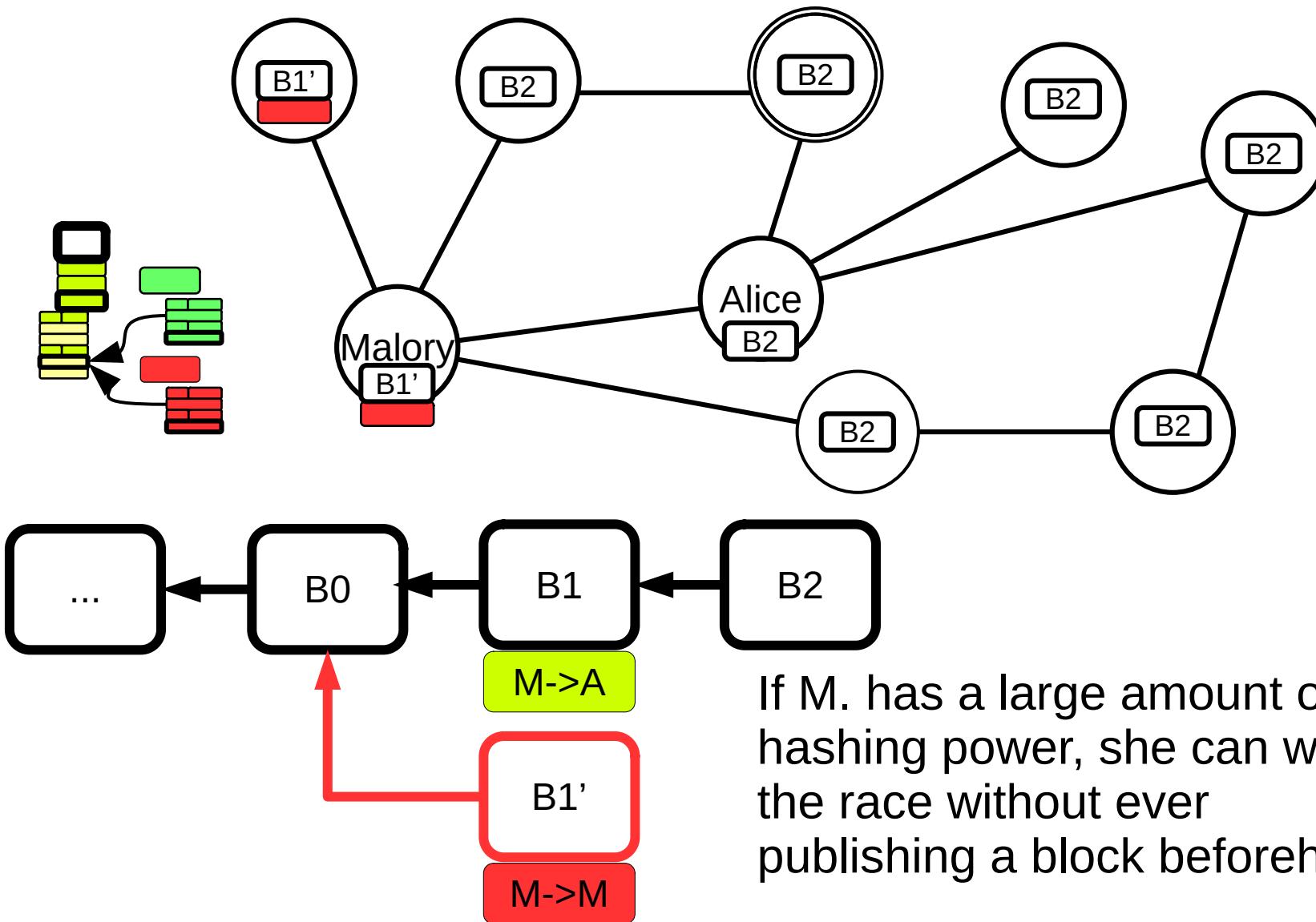
Double spending in Bitcoin



Double spending in Bitcoin

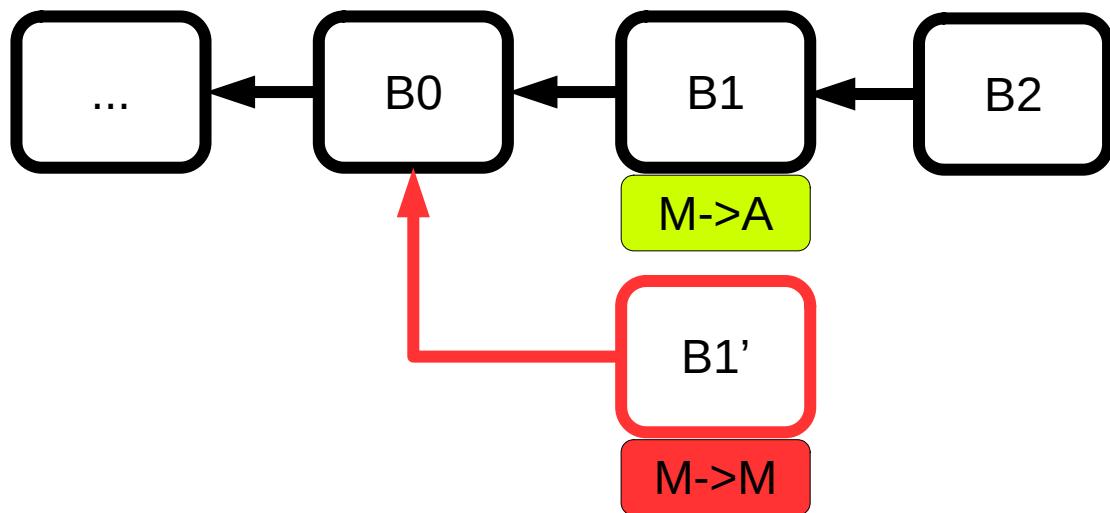


Double spending in Bitcoin



Double spending in Bitcoin

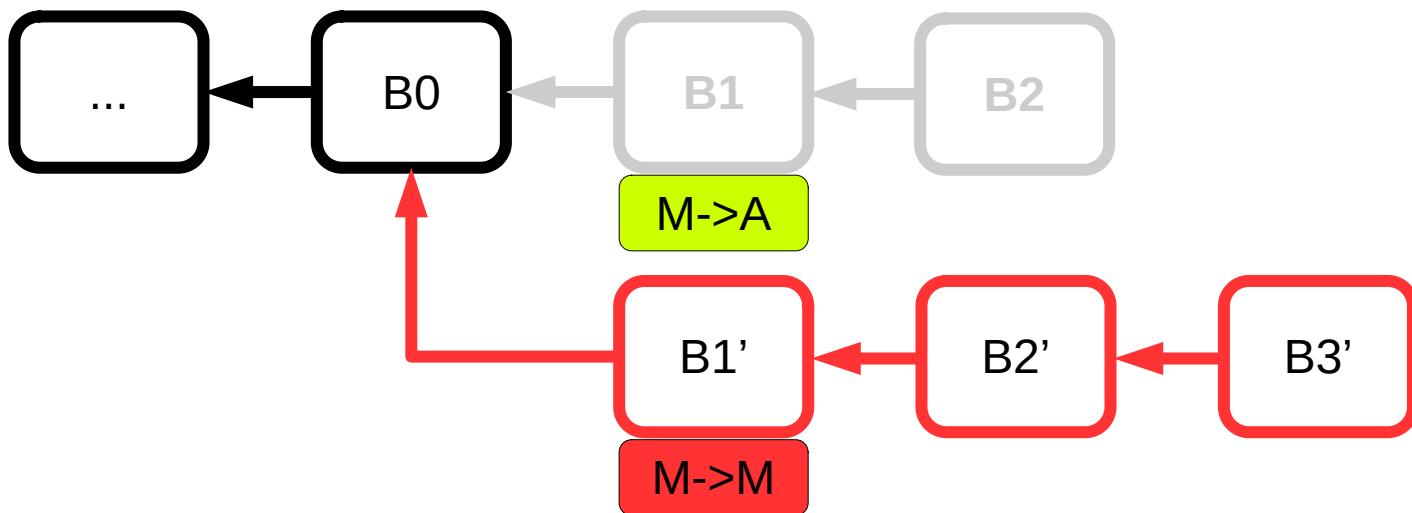
Mining a secret chain, e.g. block withholding attacks



Double spending in Bitcoin

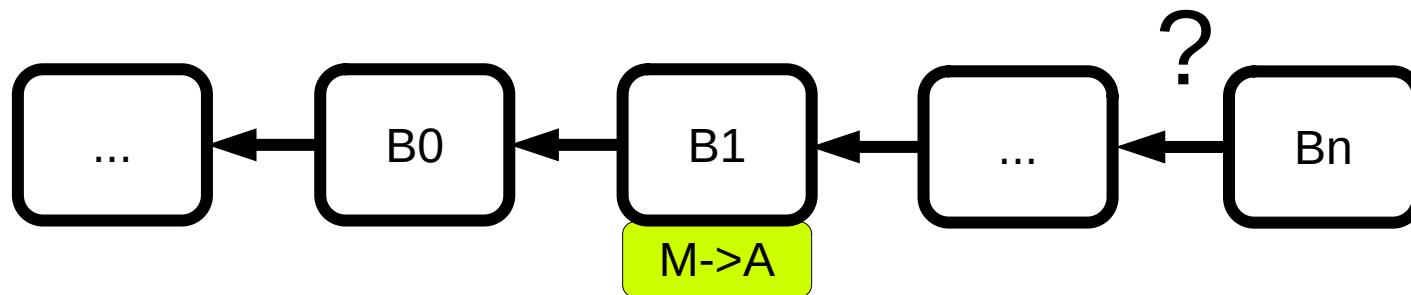
Mining a secret chain, e.g. block withholding attacks

- Requires a lot of hash rate to reliably pull-off attack
- If an attacker can choose when to do the attack
it can succeed with probability 1
 - even with low hash rate but takes very long time



Double spending in Bitcoin

- Strategies against double spending attacks:
 - Detect: check for contradicting transactions
 - How do you decide if this was an attack?
 - Difficult to reliably detect
 - Prevent: wait for confirmation blocks.
 - How many confirmation blocks are required to be on the safe side?



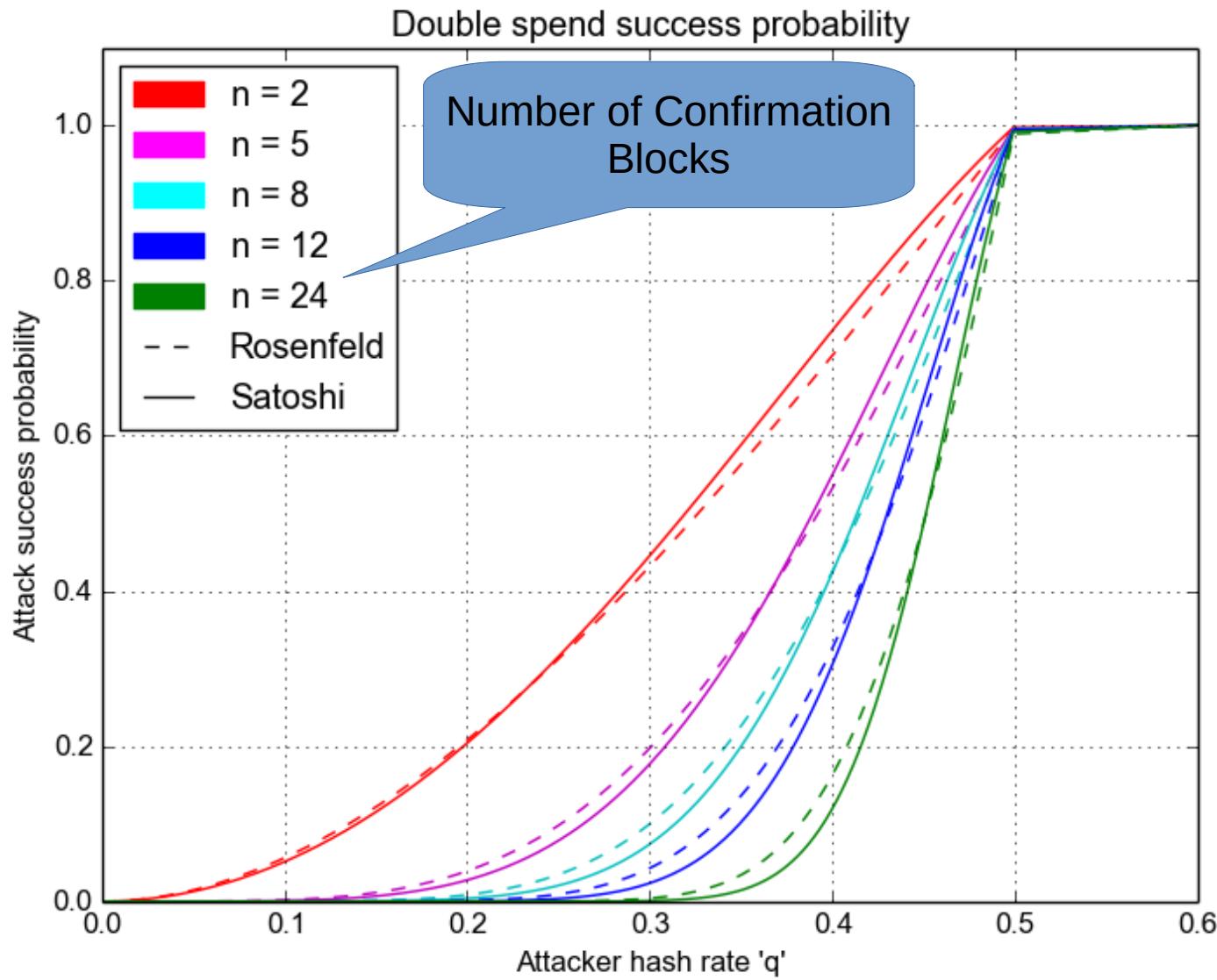
Double spending in Bitcoin

- A race between the benign and the fraudulent blockchain
- There are different models and attacks
- A simple approximation for this race is a *binomial random walk* [1,2]
 - Under the assumptions:
 - The *hash rate* of the attacker and the honest network remains constant
 - The *difficulty* remains constant

[1] [nakamoto2008bitcoin]

[2] [rosenfeld2014doublespending]

Attack success probability



[rosenfeld2014doublepending]

Double spending in Bitcoin

- Observations
 - The security in Bitcoin is based on a *probabilistic system*, i.e. no amount of confirmations will result in a attacker success rate of 0
 - The probability of a successful attack *decreases exponentially with the number of confirmation blocks*. The rate of the decay *depends on the attackers hash rate*.

Recap: How Blockchains Work

- **Early Cryptocurrencies** mostly relied on a **Trusted Third Party** to maintain a consistent ledger of Tokens
- Bitcoin was first to replace this Trusted Third Party through **Blockchain consensus**
- The system reaches eventual agreement on this ledger as long as a majority* of miners act honestly

*it may need to be closer to a supermajority, >75% [1]

Permissioned vs Permissionless

- - Membership
- - More Trust
- + Faster
- + Efficient
- Could be private or public
- Mostly Based on BFT protocols

- + Open access
- + Minimal Trust
- - Slower
- - Costly (Power)
- Could be private or public (Mostly public)
- Mostly PoW

PoW vs BFT

	PoW blockchain	BFT blockchain
Node identity:	open/permissionless, entirely decentralized	permissoned, nodes need to know IDs of eachother
Consensus finality:	no (probabilistic)	yes
Scalability: (no. of nodes)	excellent (thousands of nodes)	limited, not well explored (tested only up to $n \leq 20$)
Scalability: (no. or clients)	excellent (thousands of clients)	excellent (thousands of clients)
Performance (throughput)	Limited, due to possible chain forks (~ 7 to 10 tx/sec)	excellent (tens of thousands tx/sec)
Performance (latency)	high latency, due to no. of confirmations (~ 10 to 60 min plus)	excellent (matches network latency)

PoW vs BFT

	PoW blockchain	BFT blockchain
Power consumption:	high (PoW requires huge number of computations)	Standard
Tolerated power of adversary:	computing power < 50 % possible < 38.2 % one-block fork < 25 % BTC selfish mining	< 33% voting power
Network synchrony Assumption:	<i>Partial-synchrony</i> Physical clock timestamps (e.g. for block validity)	None for consensus safety (synchrony needed for liveness)
Correctness proofs:	In the honest majority model ongoing research	yes