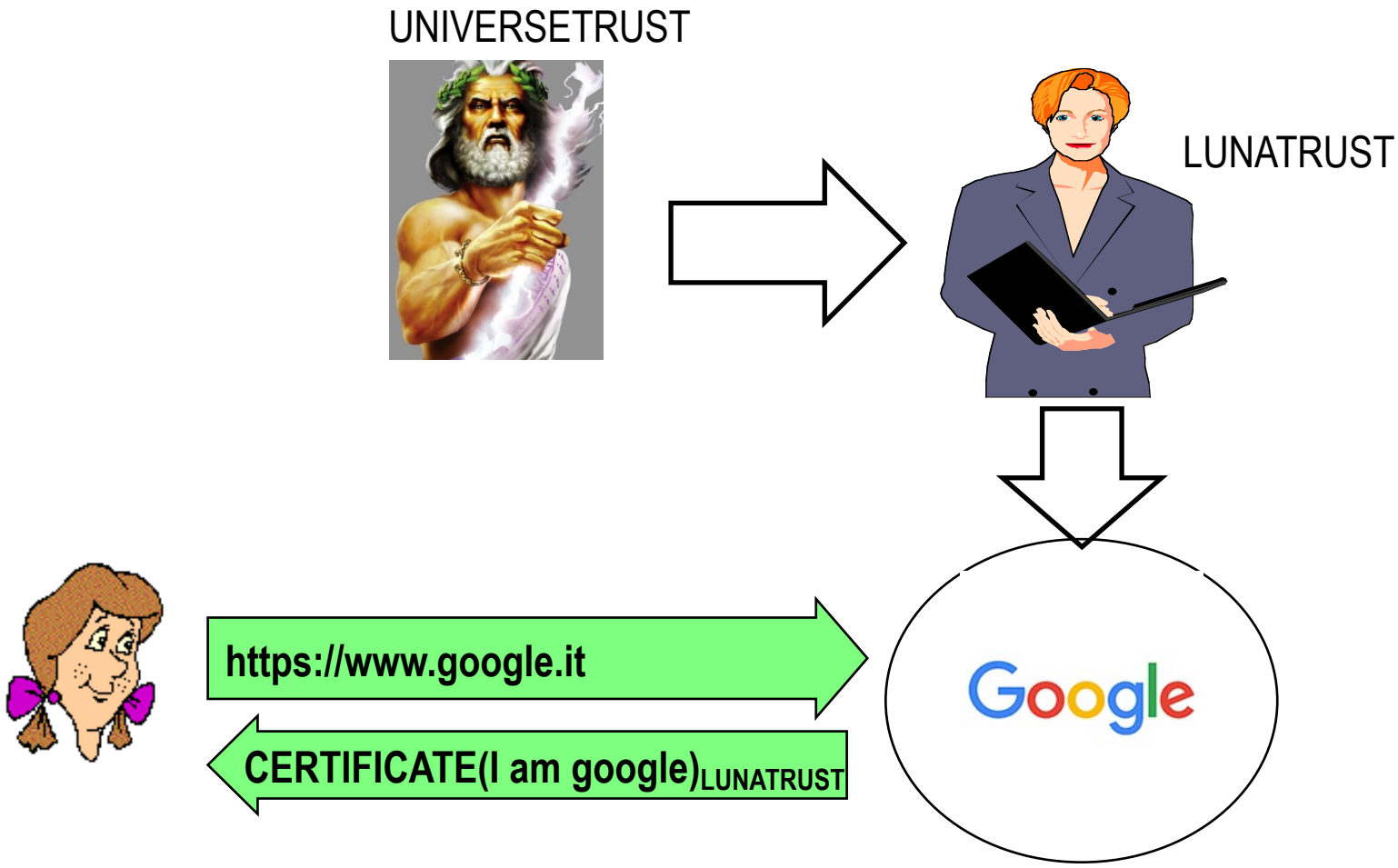
The failure of PKI
→ Certificate Transparency

(with insights on hash based data structures, specifically Merkle trees)

a VERY SERIOUS real world problem: fake certificates

Web security pillar: Certificate Authorities ARE trusted!



Fact: trusted CA assumption at stake

Fake SSL ce
threats, say

by George Leopold
Published: 09 Jan 2015

Sections

NETCRAFT

Home | News | Anti-Phishing

Fake SSL certific

Netcraft has found dozens of fa
Some of these certificates may
customers. Successful attacks
and forwarding it to the bank.
authentication credentials, or r

The fake certificates bear com
As the certificates are not sign

Online banking apps for mobile
is far from trivial, and mobile
of iOS-based banking apps test
authenticity of SSL certificates
manual tests by Leibniz Univer
may also be vulnerable if a us

Google | Online Security Blog

The latest news and insights from Google on security and safety on the Internet

Enhancing digital certificate security

Posted: Thursday, January 3, 2013

+1 183 Tweet 300 Mi piace

Engineer

cted and blocked an unauthorized digital certificate for the "*.google.com"
ly and found the certificate was issued by an intermediate certificate authority
a Turkish certificate authority. Intermediate CA certificates carry the full
has one can use it to create a certificate for any website they wish to

certificate revocation metadata on December 25 to block that intermediate CA,
other browser vendors. TURKTRUST told us that based on our information,

Our actions add
Chrome again in
though connecti

Since our priority
further discussion and careful consideration.

- **Google's VALID fake Certificates mistakenly (?) issued**
  - by TurkTrust (2012), ANSSI France (2013), etc
- **Smaller CAs: compromised**
  - Holland: Dgnotar
  - Malaysia: DigiCert sdn. Bhd.
  - etc

TLS Proxies: Friend or Foe?

Mark O'Neill, Scott Ruoti, Kent Seamons, Daniel Zappala
Brigham Young University
Department of Computer Science
Provo, UT 84602

A serious problem

- **Fake VALID certificates**
  - Variuous small/moderate CA «releasing» valid certificates for major sites
    - Asked by governments? Mistakenly… lost?
    - Many cases:
      google fake certificates issued by
      - Turkey's Turktrust, December 2012
      - ANSSI in France on December 2013
      - etc
- **Conmpromised CAs**
  - Holland: DGnotar
  - Malaysia: DigiCert sdn. Bhd.
  - etc

With powerful threats (governments), and many players which can «make mistakes» the security of the PKI model is getting weaker and weaker

How to cope with malicious CAs?

Idea: gigantic worldwide DB which anyone can check!

Certificate Transparency DB

UNIVERSETRUST

LUNATRUST

Google

They are using my name!

- non metto certificato in DB? non mi fido
- lo metto? Tutti lo vedono, e Google lo toglie

1) validation certificate
2) proof SK
3) check validation in DB

Fake!!

https://www.google.it

CERTIFICATE(I am google)LUNATRUST

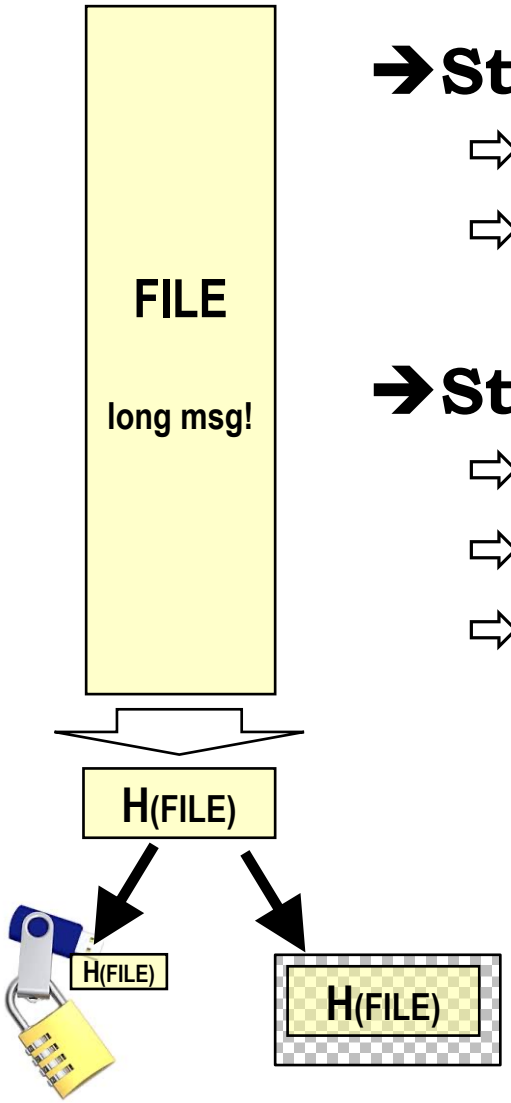
ok, c'è universetrust

How to implement such gigantic Database of certificates?

A parethesis: Merkle Trees

How to secure a file

(when you don't have a large secure storage available)



➔**Step 1: make "fingerprint" of the file**

- ⇨Via a cryptographic hash function
- ⇨large file → small fingerprint!

➔**Step 2: "secure" the fingerprint!**

- ⇨By copying it in a secure storage
- ⇨… or…
- ⇨By digitally signing the hash fingerprint

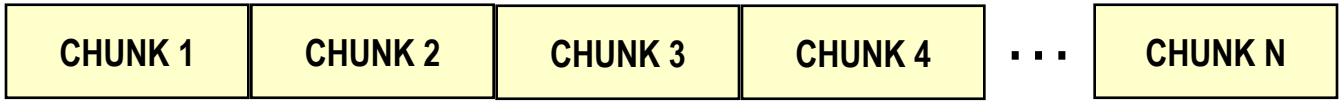
(divido in più pezzi e dato a persone diverse)

What about «chunked» messages?

➔**Peer-to-peer delivery**

⇨Message divided into independent chunks

⇨Frequently received out of order, and from distinct peers

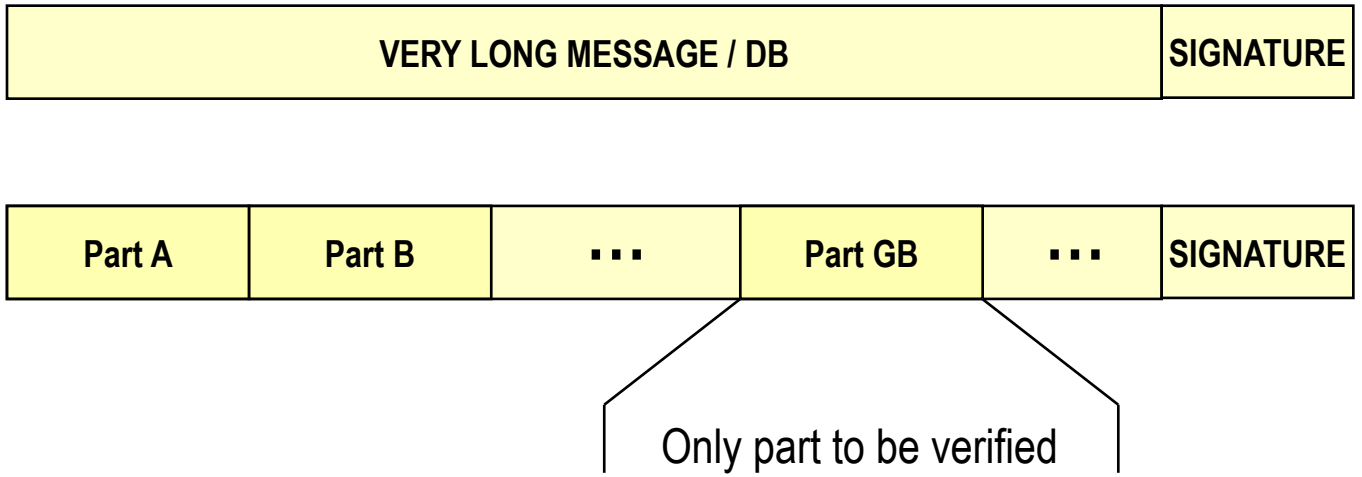


➔**Signature verification: needs to wait until COMPLETE message reconstruction**

⇨But what about fake injected chunks?

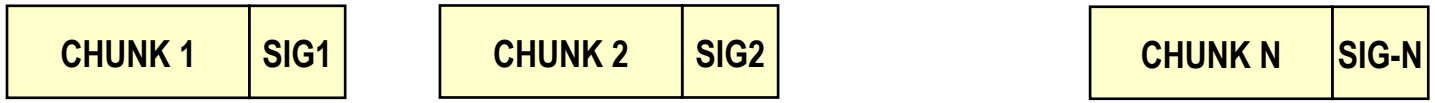
capisco solo alla fine

What about «partial» verification?



Signature verification:
need to retrieve the whole database??

Per-chunk signatures?



➔ **OK but…**

➔ **Computational overhead**

⇨ Signature = anymmetric crypto = o(1000+) x hash computation cost

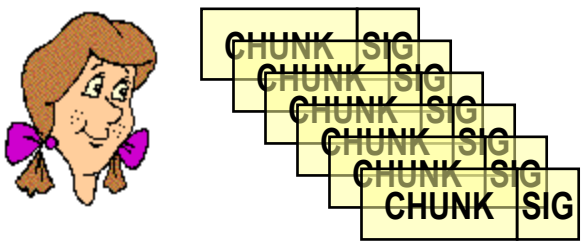
➔ **Storage overhead**

⇨ RSA2048 = 256 bytes

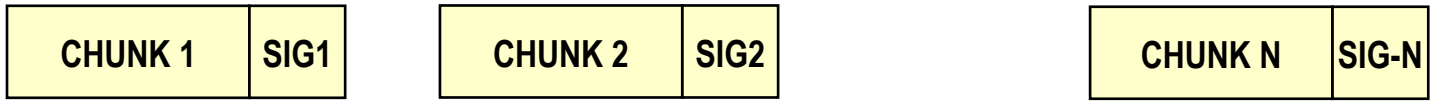
⇨ Bitcoin's ECDSA256 = 2 x 256 bit = 64 bytes

→still quite a lot f you need MANY of them (small chunks)

➔ **No more integrity check for whole file (strip-type attacks)**



Per-chunk signatures?



➔ **OK but…**

➔ **Computational overhead**

⇨ Signature = anymmetric crypto = o(1000+) x hash computation cost

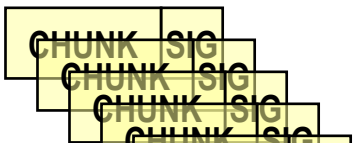
➔ **Storage overhead**

⇨ RSA2048 = 256 bytes

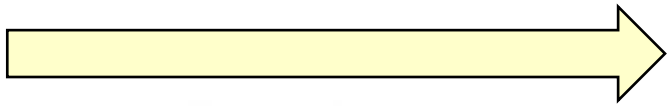
⇨ Bitcoin's ECDSA256 = 2 x 256 bit = 64 bytes

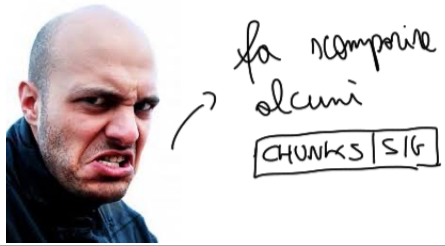
→still quite a lot f you need MANY of them (small chunks)

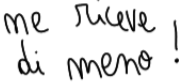
➔ **No more integrity check for whole file (strip-type attacks)**



Can we find a better solution?



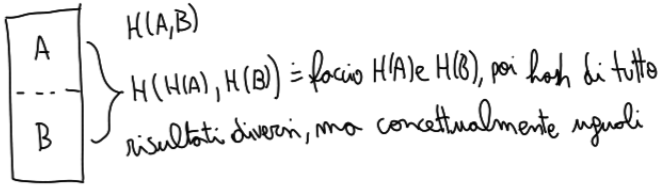




Ops… did I receive all?

SOLUZIONE

Merkle's idea: Hash tree (1979)



se cambio ordine, o tolgo un elemento, risultato cambia

Digitally signed

ROOT = H(H{H[H(A),H(B)],H[H(C),H(D)]} , H{H[H(E),H(F)],H[H(G),H(H)]})

H{H[H(A),H(B)],H[H(C),H(D)]}

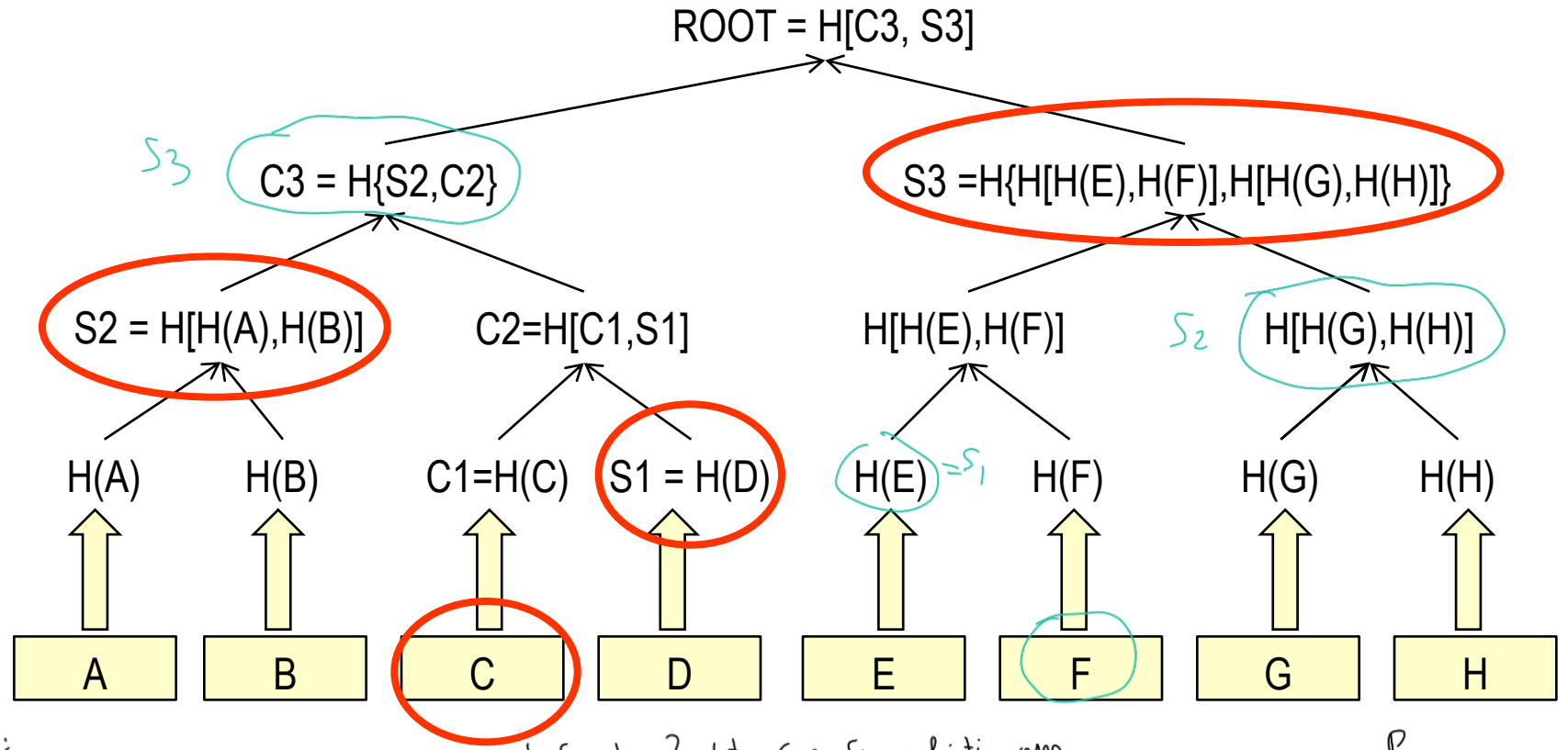
H{H[H(E),H(F)],H[H(G),H(H)]}

H[H(A),H(B)] H[H(C),H(D)] H[H(E),H(F)] H[H(G),H(H)]

H(A) H(B) H(C) H(D) H(E) H(F) H(G) H(H)

A B C D E F G H

Single chunk verification: use "siblings"



dato C:
1) H(C)
2) $H(H(C), S_1) = C_2$
3) $H(S_2, C_2) = C_3$
4) $H(C_3, S_3) \stackrel{?}{\equiv}$ ROOT → sì, ok / → no, errore nel DB

C è modificato? dato C_i e S_j cerchiati, ora ricompilare!!

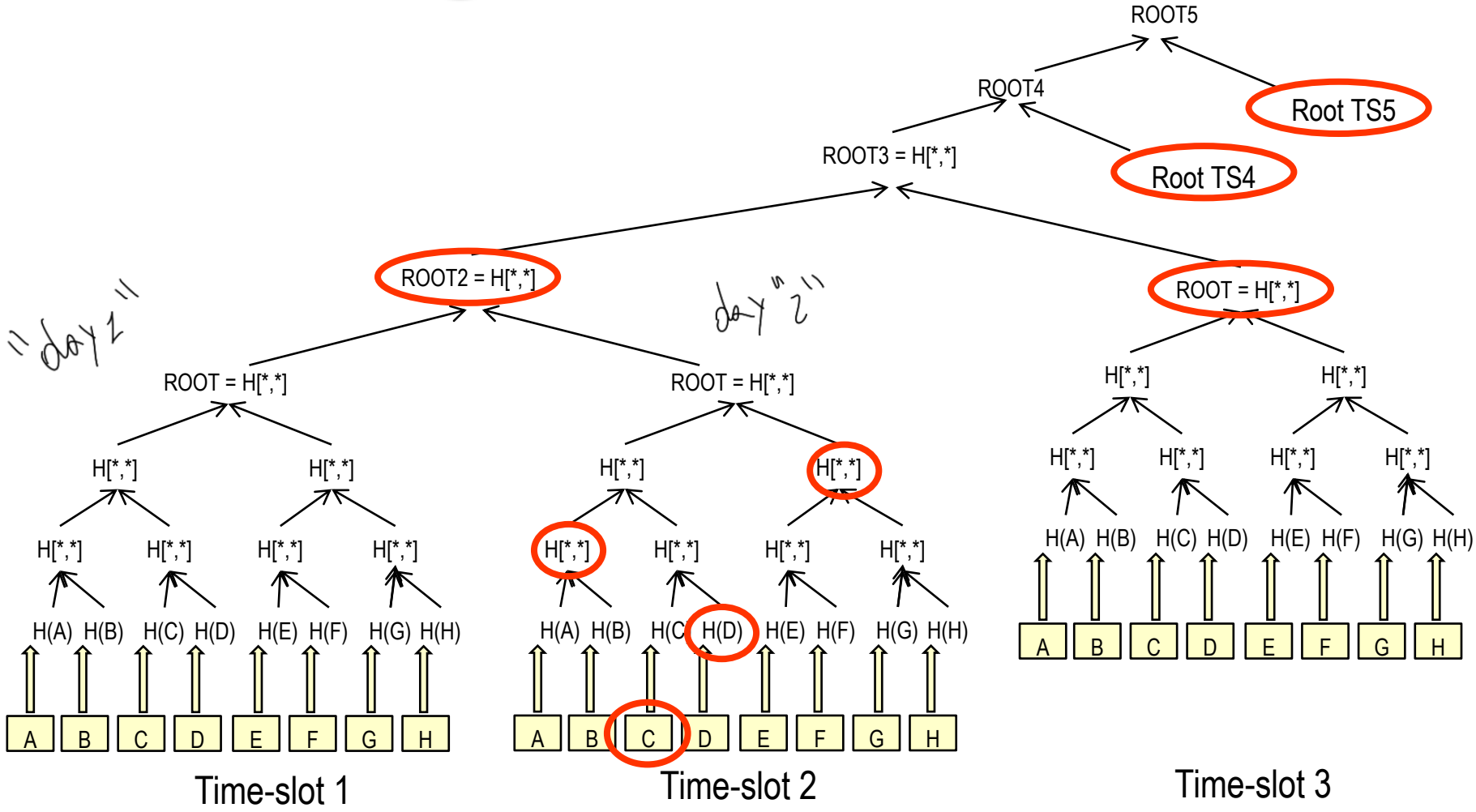
Log(N) siblings needed

per alterare C con C', dovrei generare S_i', S_i' ... che va contro le proprietà della crypto Hash

P

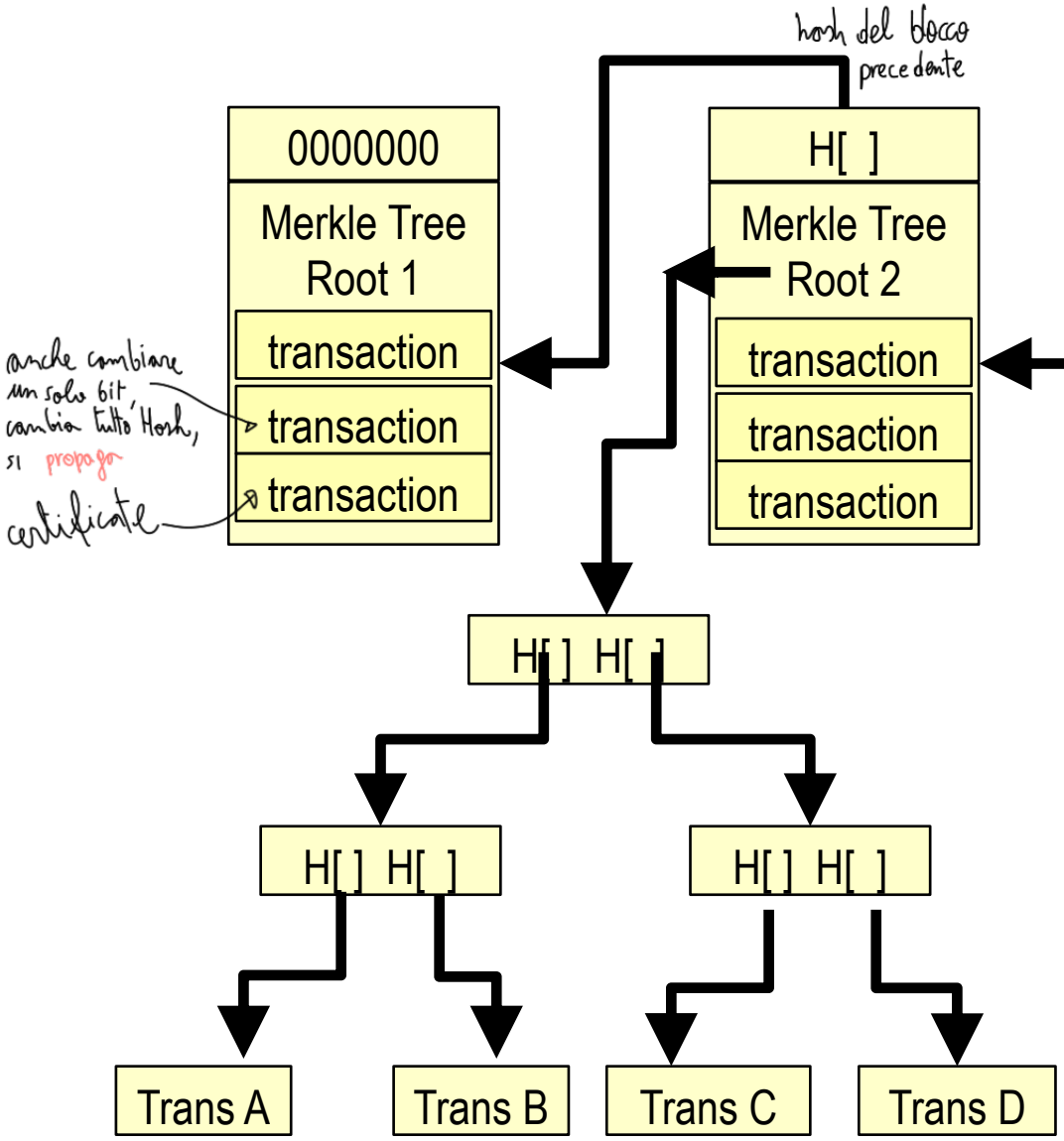
* esempio con F

Extending merkle's trees w. time



VALIDATION: Siblings for «your» tree + root ot previous timeslot + roots ot next timeslots

Merkle trees and blockchains



Transaction can be verified without downloading content of block

[Merkle tree root stored in block header]

Merkle Trees: Applications

➔**Efficient storage in blockchains**

- ⇨Bitcoin, Ethereum, etc
- ⇨Used in many parts of many blockchains!

➔**Google's Certificate transparency**

➔**File Chunk validation**

➔**And many more**

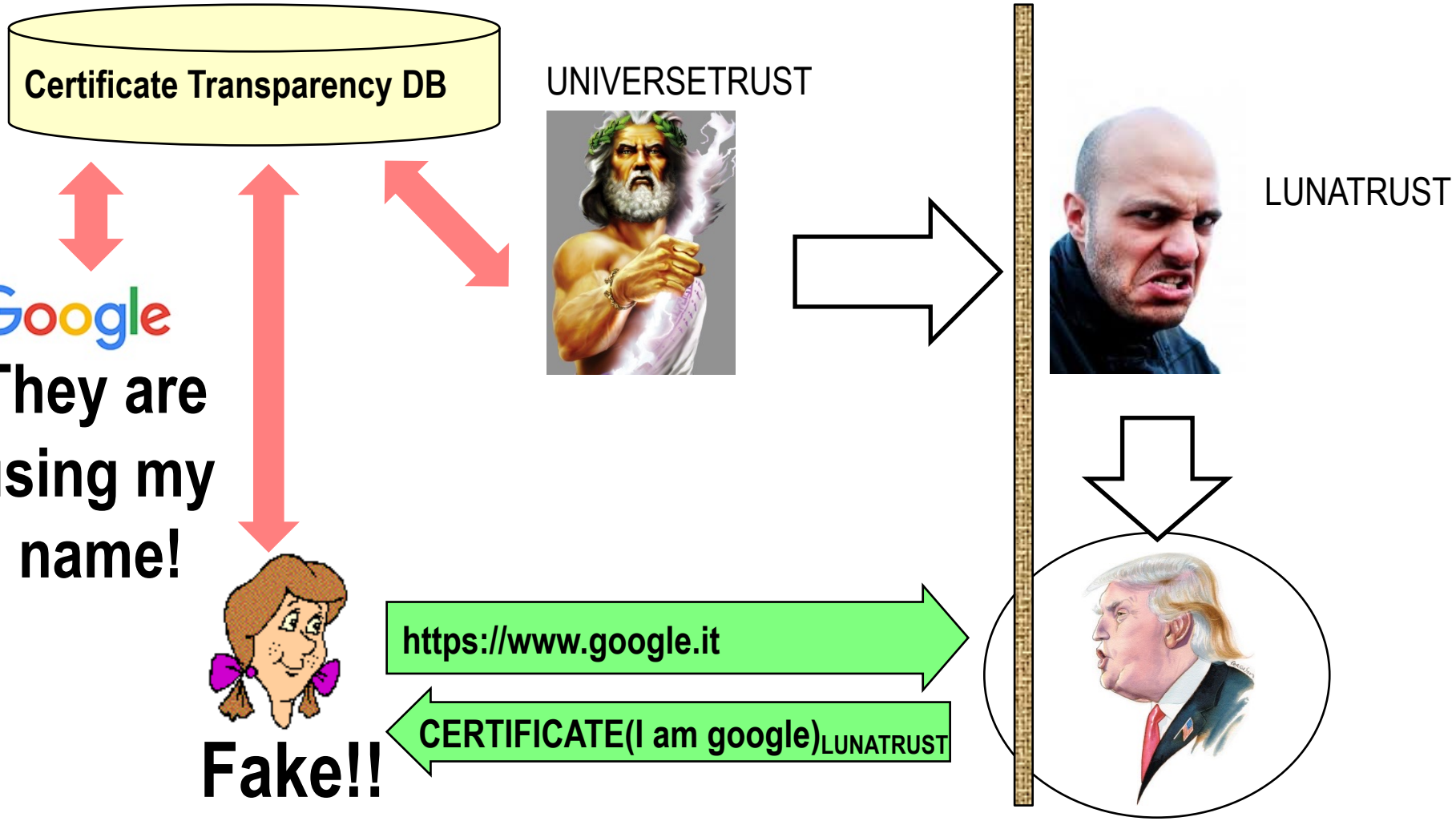
- ⇨One-time signatures
- ⇨Node authentication
- ⇨Memory & storage authentication
- ⇨Signature spreading
- ⇨…. Etc etc …

Merkle Trees' application:

Certificate Transparency

How to cope with malicious CAs?

Idea: gigantic worldwide DB which anyone can check!

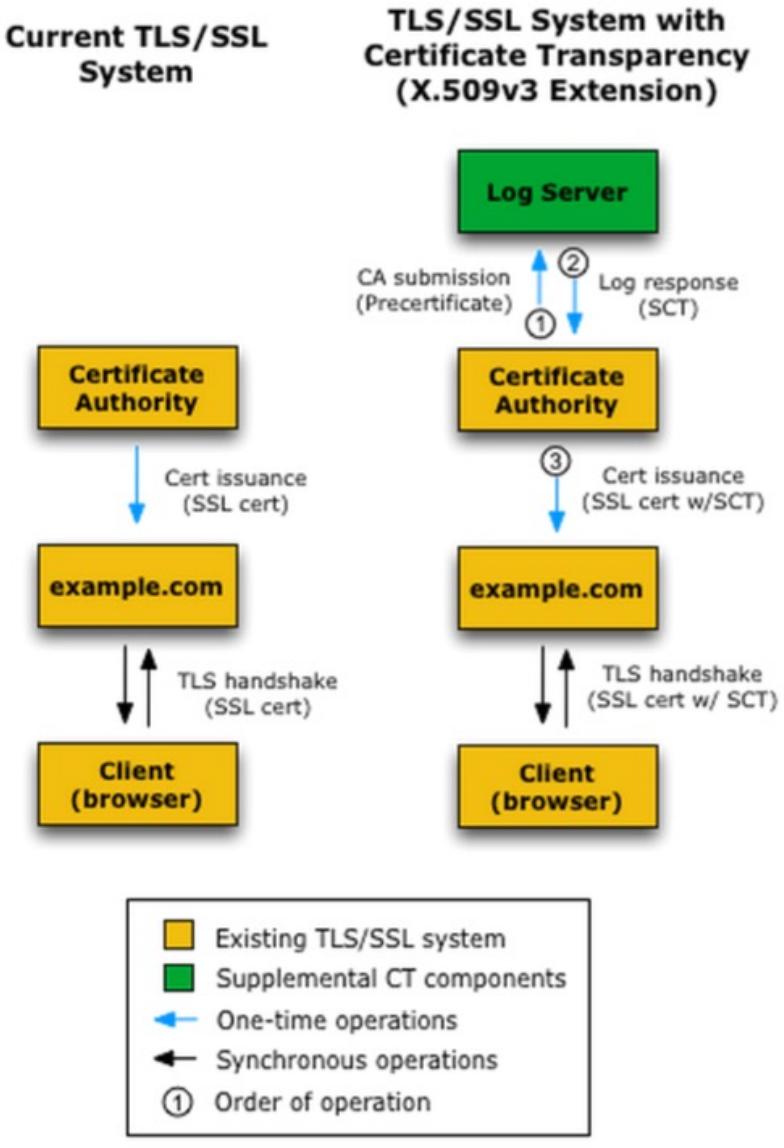


Why certificate transparency?

- **Make it almost impossible for a CA to issue a SSL certificate for a domain without the certificate being visible to the owner of that domain.**
- **Idea: put all certificates in a gigantic public list, that everyone can access and consult**
- **Further idea: make sure that we all see the "same" list!!**
  - New threat: "split world": the public list you see is different (and crafted!) from what the rest of the world sees

Certificate Transparency in a nutshell

- **When issuing certificate: add it to the log server**
  - ⇨ Returns a Signed Certificate Timestamp (SCT)
- **During TLS handshake:**
  - ⇨ Check certificate validity (as usual)
  - ⇨ Follow up with an **additional** Extended Validation (EV) by verifying that the certificate is in the public list



Certificate Transparency

- **Google's initiative**
  - started 2013
- **First stated in experimental RFC 6962**
  - July 2013
- **Endorsed and supported by IETF WG «trans»**
  - Goal: standardize CT
  - RFC 6962bis released in December 2014
- **Currently integrated in major browsers**
  - Chrome 1+, IE7+, Firefox 3+, Safari 3.2+, Opera 9.5+
  - Although not yet consistently used by all web sites
  - Plan to improve support for Extended Validation by 2015
- **Details @ http://www.certificate-transparency.org/**

CT: how it works?

- **Uses Merkle trees!**
  - Very scalable, log(n) verification complexity
- **A special type (sometimes called chron tree)**
  - «append only» Merkle tree
  - Timestamped entries
    - Cannot retroactively insert a certificate

Following figures: source http://www.certificate-transparency.org/log-proofs-work

CT: how it works?

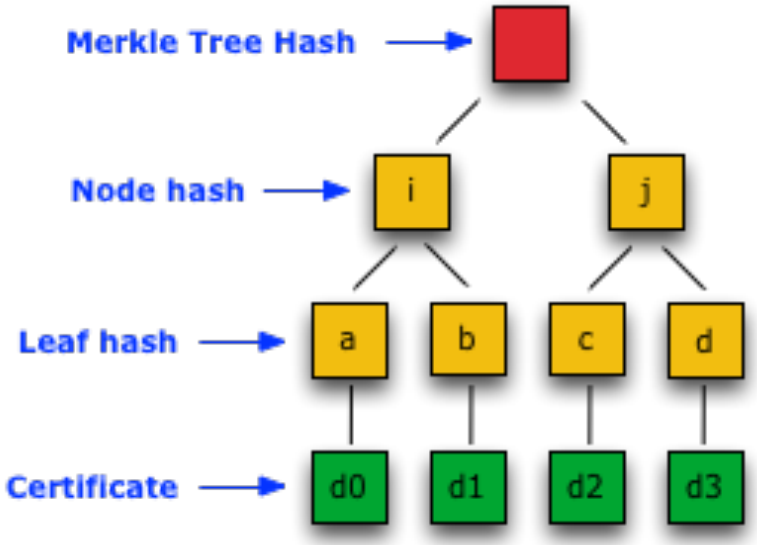


Figure 1

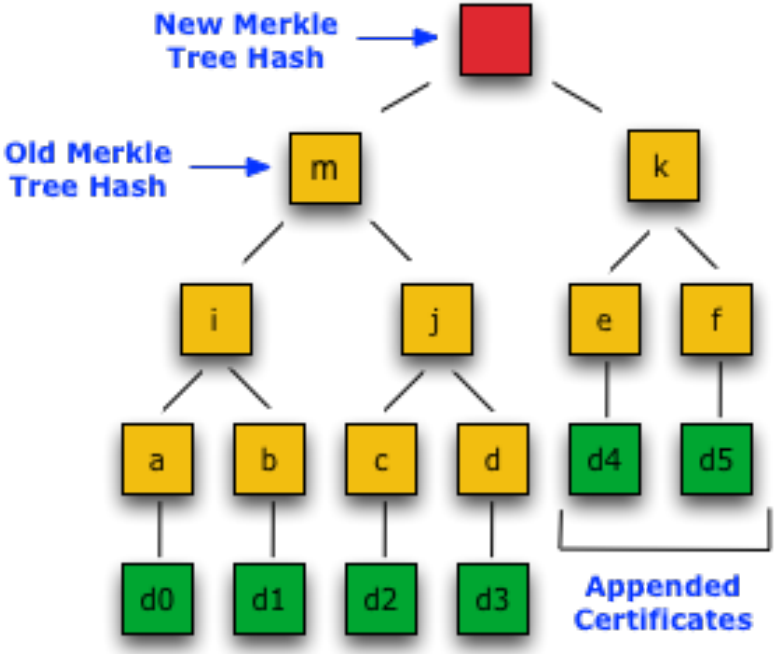


Figure 2

Every X hours, tree updated (currently X=24h)

Merkle Consistency proof

Verify that two versions of a log are consistent (i.e., the later includes everything of the earlier).
Consistency proof → no certificates have been modified or back-dated and inserted into the log.
Proof = need only root of the subtrees – includes past "root"

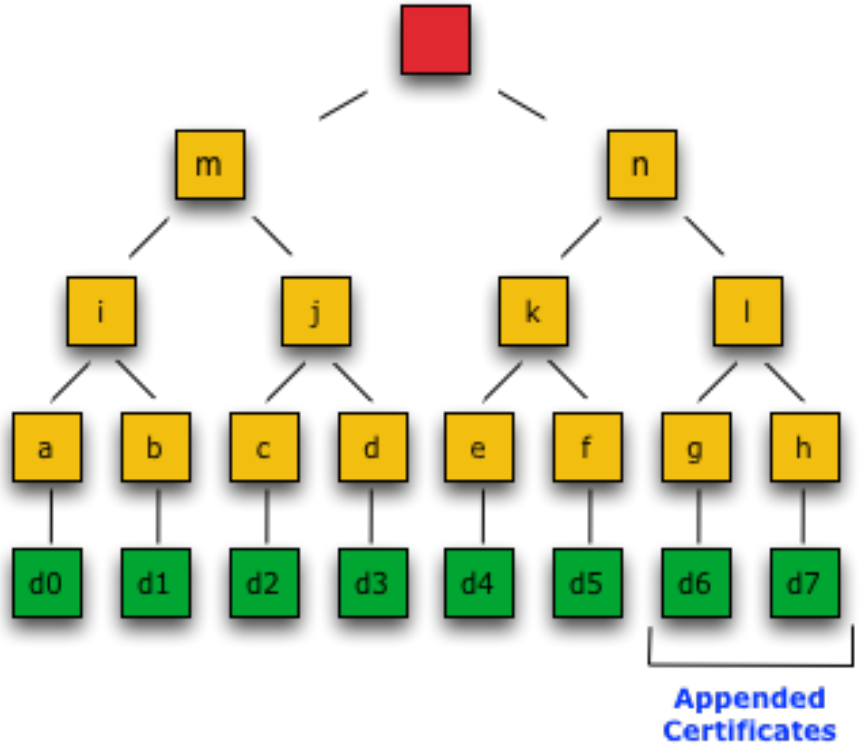


Figure 3

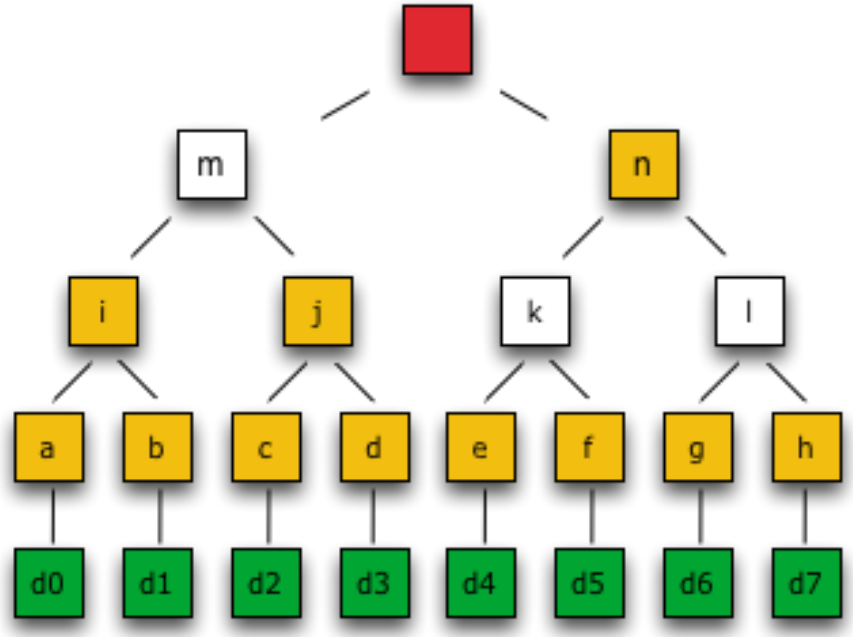


Figure 4

Merkle Audit Proof

- **Usual: via certificate + siblings**
- **Siblings obtained from log**
  - Issue: online operation, visible to all…
  - Remember who are the threats today…
- **Alternative: use «gossiping» protocols**
  - Work in progress @ 2015!
  - No time for (very nice) details…

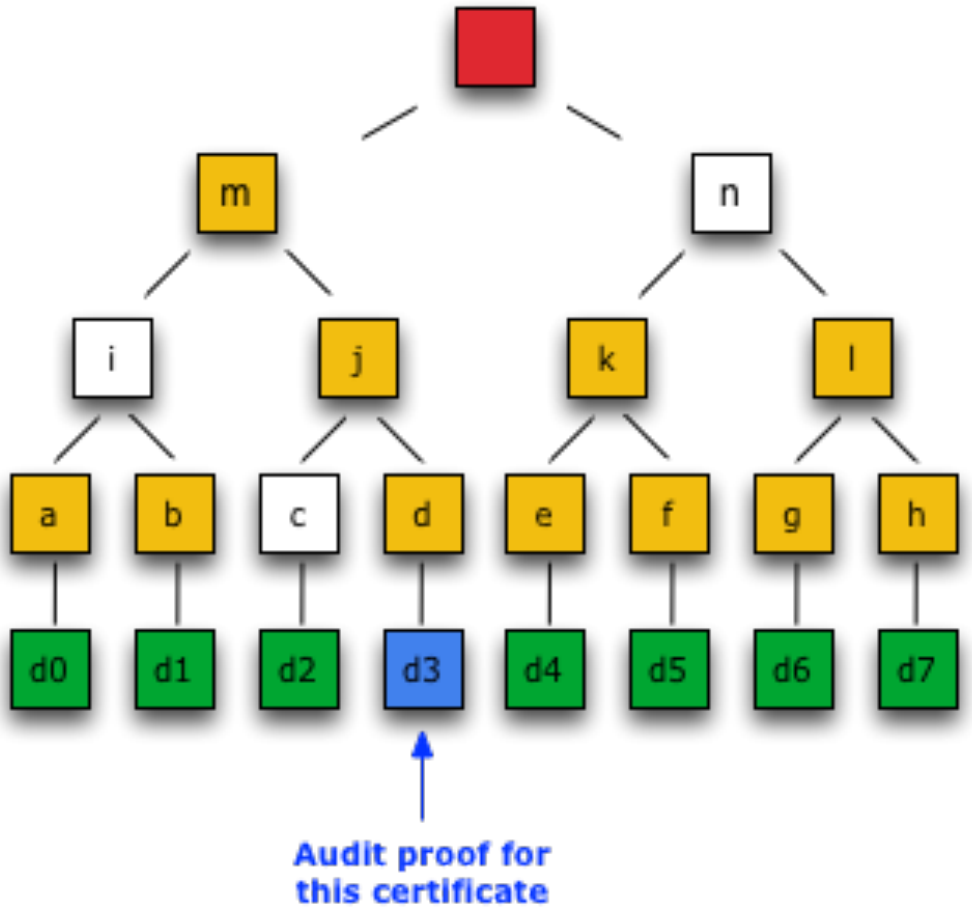


Figure 5

EV in the real world