Technische Universität Berlin

EINSTEIN CENTER Digital Future

TIMESTAMPING

Blockchain Technologies | Summer 2019

Prof. Dr. Florian Tschorsch
Distributed Security Infrastructures

Motivation

- timestamp the creation/modification of a data object
- use cases include intellectual property, verifiable logs, ...

- in the real world, a notary can provide a timestamping service
- for example, maintain a diary with dated entries, no pages left blank, and regular stamps by a notary
  - sewn-in pages of the notebook make it difficult to tamper
- mail a letter to oneself, leave it unopened; ensures that the enclosed letter was created before the time postmarked on the envelope

- two hidden assumptions:
  - signs of tampering are visible
  - trusted third party exists

Naive Solution

- a naive solution would be a “digital safety deposit box” managed by a timestamping authority (TSA)
  - client sends data object to TSA
  - TSA records a timestamp and a copy of the data
  - data integrity can be challenged by comparison
  - timestamps are managed by the TSA
- this solution has several issues
  - privacy
  - bandwidth and storage
  - trust

We can address the first two issues pretty easily (see the following). Solving the trust issue, however, is difficult!

Hash Functions

- hash functions take any input and produce a fixed-size *digest* as output
- they are deterministic, i. e., the same input always returns the same output
- common (cryptographic) hash functions, e. g., SHA2, SHA3, or Blake2, produce digests ranging from 256 bits to 512 bits
- a function $H(\cdot)$ is considered a *cryptographic hash*, if it meets some specific security requirements
- in the following, we focus on three requirements
  - pre-image resistance
  - second-preimage resistance
  - collision resistance
- the example hash functions above are believed to provide all of these properties
  - nobody has articulated a meaningful attack but could always change

Hash Functions: Pre-image Resistance

- given some output $y = H(x)$, it should be infeasible/difficult/time-consuming/... to find an input x with
$$H(x) = y$$
- ideally the best attack should require a brute-force search
- also referred to when it is said that H is a *one-way function*

- related is the so-called hiding property
- a hash function H is *hiding*, if it is infeasible to find x for $H(x \mid\mid r)$, where r is a secret random value
- we can use this to hide the value x, even if it is a binary value, e. g., the result of a coin toss
- the application domain are so-called commitment schemes

Hash Functions: Second-preimage Resistance

- this is subtly different than pre-image resistance
- assume some input x is known and "fixed" in advance
- given x, it should be infeasible to find a *different* input x' such that

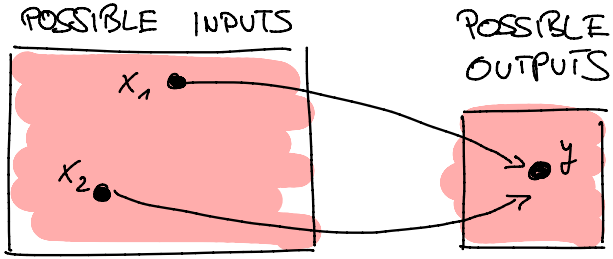
$$H(x) = H(x')$$

Hash Functions: Collision Resistance

- it should be infeasible to find *any* two values x_1, x_2 such that

$$H(x_1) = H(x_2)$$

- note that this is a much stronger assumption than second-preimage resistance, since the attacker has complete freedom to find a collision



Signature Schemes

- a user (or signer) generates a pair of keys, called the public key *pk* and private/secret key *sk*
- that is, we use asymmetric cryptography
- the user can use *sk* to "sign" messages, i. e., produce a digital signature *sig*

$$sig = \texttt{sign}(sk, \texttt{message})$$

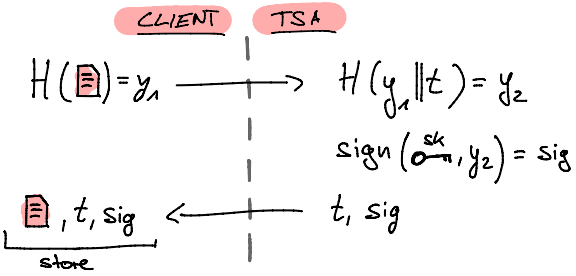
- anyone in possession of *pk* can verify the correctness of a message and its signature

$$\texttt{verify}(pk, \texttt{message}, sig) == \texttt{true}$$

- we require unforgeability, i. e., without *sk* nobody should be able to provide a valid signature

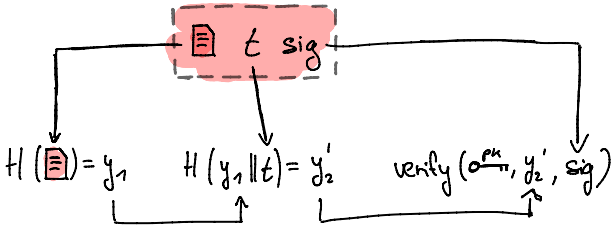
Trusted Timestamping: Creating a Timestamp

- client hashes the data object
  - the hash serves as a digital fingerprint of the original data
  - if the data is changed, this will result in a completely different hash
- send the hash to the TSA
  - confidentiality: original data cannot be calculated from the hash
- TSA concatenates a timestamp, calculates a hash, and signs it
- TSA sends the signature and the timestamp to the client, who stores these information with the original data



Trusted Timestamping: Checking a Timestamp

- the hash of the original data is calculated, the timestamp is appended, and the hash of this concatenation is calculated
- verify the signature by taking the TSA's *pk*
  - proves that the timestamp and the message are unaltered
  - proves that the timestamp was issued by the TSA



Trusted Timestamping

What is the major problem with the trusted timestamping approach?

- we successfully addressed privacy and storage/bandwidth requirements
- yet, we still have to blindly trust the TSA
- TSA can forward date and backdate data objects

Linked Timestamping

- observation: the sequence of clients requesting timestamps and the hashes they submit are not predictable
- idea: include information from previous clients in the signature
- temporal order of issued timestamps is protected
- modifications of issued timestamps would invalidate this structure

- TSA issues signed sequentially numbered timestamp certificates C_n

$$C_n = (n, t_n, ID_n, y_n, L_n) \qquad sig = \mathtt{sign}(sk_{TSA}, C_n)$$

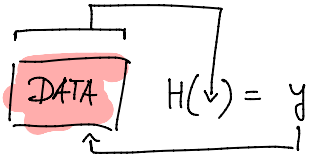
with sequence number n, time t_n, client identifier ID_n, the data hash y_n, and *linking information* $L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, H(L_{n-1}))$

Haber and Stornetta. “How to time-stamp a digital document.” 1990.

Hash Pointer

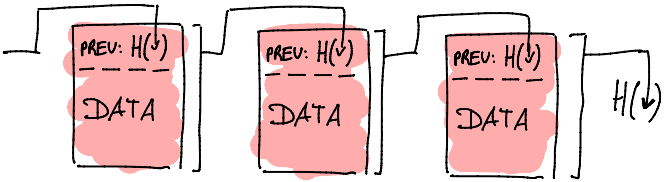
- a hash pointer is a data structure that can be thought as a regular pointer including a cryptographic hash
- a regular pointer provides a way to retrieve information
- a hash pointer also provides data integrity

- basically, we can use hash pointers to augment data structures that use pointers, e. g., a linked list or a binary search tree



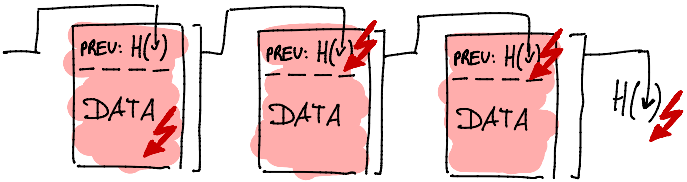
Hash Chain

- observation: C_n contains information immediately preceding the desired time
- makes forward dating difficult, because the preceding information might be lacking



Hash Chain

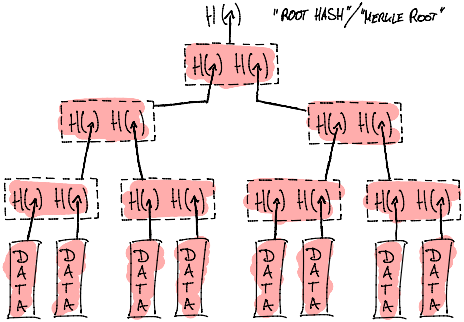
- observation: C_n contains information immediately preceding the desired time
- makes forward dating difficult, because the preceding information might be lacking
- makes backdating difficult, because certificates have already been issued
  - correctly embedding a forged certificate into the already existing stream of timestamps requires the computation of a hash collision
- the only possible spoof is to prepare a fake chain of timestamps
  (long enough to exhaust the most suspicious challenger, i. e., up to the beginning)



Hash Tree

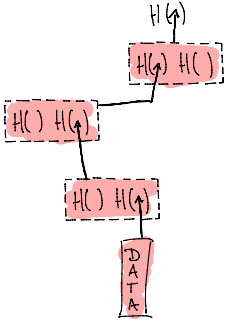
- a binary tree with hash pointers is known as a *Merkle tree*
- data objects comprise the leaves arranged in pairs of two
- for each pair, we build a node that has two hash pointers, one to each child node
- we repeat the procedure on this new level of nodes and continue until we reach the root

- we only remember the so-called *Merkle root*, i. e., the root hash
- tampering with some data will cause that the respective hash pointers will not match, eventually propagating even to the root



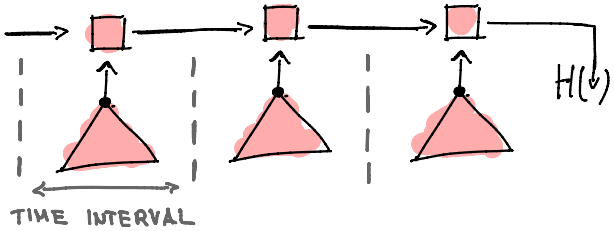
Hash Tree: Proof of Membership

- unlike the hash chain, hash trees allow a concise *proof of membership*
- in order to prove that a certain data object is a member of the Merkle tree
  - show the data object and the nodes on the path from the data block to the root
  - we can ignore the rest of the tree, as the data objects are “included” in the hash pointers
- if there are n nodes in the tree, only about $log(n)$ nodes need to be shown/verified



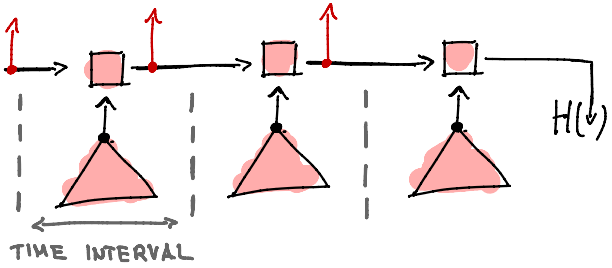
Hash Chains meet Hash Trees

- extension: combine hash chains with Merkle trees,
  which makes timestamping more efficient
- introduce time intervals to reduce complexity even further
  (depending on the implementation it might reduce time resolution)



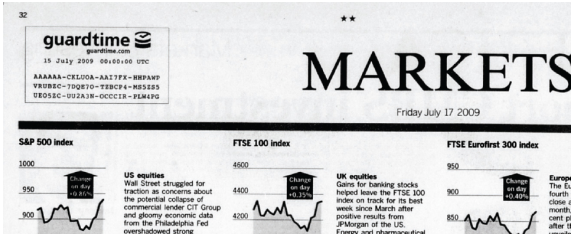
Trust Issue

- to some extent, we can mitigate the trust issue:
  TSA periodically publishes some hash pointers
- by publishing *witnessed links*, the TSA creates unforgeable verification points
- we still need a trusted third party (TTP)



Trust Issue

- to some extent, we can mitigate the trust issue:
  TSA periodically publishes some hash pointers
- by publishing *witnessed links*, the TSA creates unforgeable verification points
- we still need a trusted third party (TTP)


32

guardtime
guardtime.com
15 July 2009 08:05:00 UTC
AAAAAA-CELUOA-AAI7FX-HHPAWP
VRUBEC-7PQETO-YEBCP4-MSSZES
UEO5IC-OU2AJN-OCCCIR-PLM4PG

MARKETS

Friday July 17 2009

S&P 500 index
1000
950
900
Change on day +0.85%

US equities
Wall Street struggled for traction as concerns about the potential collapse of commercial lender CIT Group and gloomy economic data from the Philadelphia Fed overshadowed strong

FTSE 100 index
4600
4400
4200
Change on day +0.35%

UK equities
Gains for banking stocks helped leave the FTSE 100 index on track for its best week since March after positive results from JPMorgan of the US. Energy and pharmaceutical

FTSE Eurofirst 300 index
950
900
850
Change on day +0.40%

Europe
The Ex
fourth
close
month
cent p
after t

Distributed Trust

- idea: in order to get rid of the TTP, let us try to “distribute the trust” by asking multiple nodes for a signature
- assume a public-key infrastructure (PKI), a public directory of nodes and a pseudorandom generator G is available
- G takes an input as seed and outputs a random sequence, therefore not predictable
- client ID uses the hash value y as a seed for G to generate a k-tuple with k other node identifier ID_j

$$G(y) = (ID_1, ID_2, \ldots, ID_k)$$

Haber and Stornetta. “How to time-stamp a digital document.” 1990.

Distributed Trust (cont'd)

- client sends (y, ID) to each node in $G(y)$ and receives

$$sig_j = \texttt{sign}(sk_j, t||ID||y) \quad \text{with } j \in \{1, \ldots, k\}$$

- the timestamp now consists of $[(y, ID), (sig_1, \ldots, sig_k)]$

- the only way to produce a forged timestamp is to use a hash y so that $G(y)$ names k malicious nodes
  - if G is a secure generator, finding such a y is infeasible

- in order to cope node failures, we can relax the requirements and consider a timestamp valid if clients can provide k' signatures, where $k' < k$
  - increases the degree of freedom for adversaries

Haber and Stornetta. "How to time-stamp a digital document." 1990.

Open Discussion

What are the pros and cons of the respective approaches?

Do you see any (theoretical or practical) weaknesses?

Ask yourself, for example, where is (some) trust still necessary?

Conclusion

- timestamping suffers from a number of issues, most notable “trust”
- we have discussed two (separate) approaches,
  i. e., linked timestamping and distributed trust
- they address some of the issues
- linked timestamping makes use of hash pointers; in particular
  - hash chains
  - Merkle trees
- distributed trust gets rid of the trusted third party