Commitment-Schemes

Leonhard Applis

TH Nürnberg

XX.XX.XXXX

Table of Contents

Problem(s)

to do: What are the problems we need to adress

Commitments

- A **commits** to B
- B keeps commitment, is unable to read or process it
- A reveals to B
- B can verify the commitment

TODO: Image

Attributes

1. **Binding:** The Values Alice put in the Commitment cannot be changed after B recieved it
2. **Hiding:** Bob cannot gain any information about the message from the commitment itself
3. **Viability:** If both parties follow the Protocoll, Bob is always able to recover the committed value

Additional for *real-life-applications*:

1. Bob's are able compare commitments
2. Commitments are *tradeable*

Applications

Challenge and Response
You can setup your own anonymus challenges, leaving a commitment at Bob's. If someone show's up saying he's Alice, Bob challenges to reveal the commitment.

JSON-Web-Tokens (JWT):
A payload (e.g. some account details) are encrypted to a commitment and passed to a third party.
You can verify yourself at the third-party revealing the commitment
this is done *automatic* via session or systemattributes

Table of Contents

1. Alice produces $h = Hash(m)$ and sends Bob h and $Hash$
2. Bob keeps h and $Hash$
3. Alice reveals herself by sending Bob m
4. Bob checks if $Hash(m) \equiv h$

Important: NEVER use actual important data as message, you send it in cleartext in Step 3.

Fullfillment of Attributes

Hiding: because of the Hash-functions **Pre-Image resistance**, it's nearly impossible to find the message m from the hash. This holds true for any Bob and any Eve.

Binding: because of the Hash-functions **collision-resistance**, it's nearly impossible to find another message m with the same hash.

Hash-Based Commitments

Problem: unlimited range - limited domain

Usually: Bob (and Eve) are not able to *guess* m from h and $Hash$

But: if the *plausible domain* of m is known, its possible for modern computers to brute force reveal your m

Example: Alice commits to Bob about the result of a soccer game Germany vs. Brazil.
Therefore she chooses a score of 0:7 and sends Bob $h = SHA_3(str(0:7))$ and the Hashfunction SHA_3
Eve catches the commitment and knows the context of the soccer game. she can know try reasonable combinations of results from 0:0 up to 20:20. She only needs to try $20 \cdot 20 = 400$ results

Improved Concept:

- Alice chooses a random value s
- Alice produces $h = Hash(m, s)$ and sends h and $Hash$ to Bob
- Bob keeps h and $Hash$
- Alice reveals herself by sending bob m and s
- Bob checks if $Hash(m, s) \equiv h$

Addition

Alice is anonymus. She never stated her name, used certificates, etc. Alice can produce as many commitments for as many personas as she wants.

For increased security:

- commitments should be one-use only
- commitments should have a lifetime
- traded commitments to a third Party should revealed directly with first reveal
- messages must be chosen random

Table of Contents

Binary-Concept
Requirements and Definitions

Table of Contents

Prerequisites: Bob needs to setup the environtment for alice, by

1. choosing a large prime number p
2. choosing a smaller prime number $q \in \{1..p | q \div (p-1) = 0\}$
3. choosing $g, v \in G_q \neq 1$
4. sending Alice p, q, g, v

Now Alice can *build* the exact same group and subgroup like Bob. This is similiar to sending the hash-function.

Implementation

- Alice requests p, q, g, v from Bob. Alice check that q, p are primes, q divides p-1, that g and v are valid elements.
- Alice chooses her message $m \in \{1..p\}$ and a random number $r \in \{1..q-1\}$
- Alice sends $c = g^r v^m$ to Bob **(commit)**
- Bob keeps $< Alice, c, < p, q, g, v >>$
- Alice can reveal herself by sending r, m to Bob. Bob checks $c = g^r v^m$

Benefits

Major:

- Commitments always contain random parts
- No collision possible (unlike Hashfunctions)

Minor:

- tupels are smaller to store than hashes
- p, q, g, v are easily changed/renewed (you could not renew hashfunctions)