Commitment Schemes

Leonhard Applis

***Abstract*—This Paper summarizes and introduces to the topic of Commitment-Schemes, a two party kryptographic protocol.**

The aim of commitment schemes is to provide a mechanism for Party A to commit to a hidden value and reveal it if necessary. Party B can confirm that the revealed value and the hidden value match.

This Paper first introduces to the topic itself, a hash-based implementation and the *pedersen-commitments* which are based on the discrete logarithm. In Conclusion a Case-study of commitments for one-time authorization in a distributed web-application is provided.

I. Introduction

The Internet is a land of mistrust for good reasons, yet it's often required to trust each other.

A common example for the use of commitment-schemes is the *coin-toss via telephone*: Alice declares her call, and Bob tosses the coin. However, Alice is unable to *see* the real coin-toss and relies on Bobs righteousness - which is a pretty bad idea.

Evil Bobs can simply declare the wrong toss for Alice' call, making her loose every single time. Overall Bob can manipulate the outcome easily by choice with his knowledge of Alice' call.

Therefore Alice needs to hide her prediction. A simple way of *saving* Alice from manipulation is that Alice declares her call *after* Bob announces the coin-toss-outcome. Needless to say, now Alice is in the position to make her calls in her favor every time, which is not acceptable for Bob.

Commitment-schemes enable both parties to a *fair-play* and tools to detect manipulation creating relative trust. In a successful commitment, Alice hides her guess with kryptographic measures and sends it to Bob. Bob then declares the coin-toss to Alice, without knowledge of her call. At this point Alice knows, if she has won or lost, and enables Bob to read her guess by sending the required keys for decryption.

At this point both parties know the *real* outcome and the real winner of the coin toss. The measures if any manipulation is detected are beyond this protocol.

There are several cases where commitment-schemes are commonly used, for example in *challenge and response*-authentication, whereof a simple example is provided at the end of this paper in section III.

Also, other higher protocols require commitments, such as *secret-sharing*. Unfortunately these protocols are out of scope for this paper.

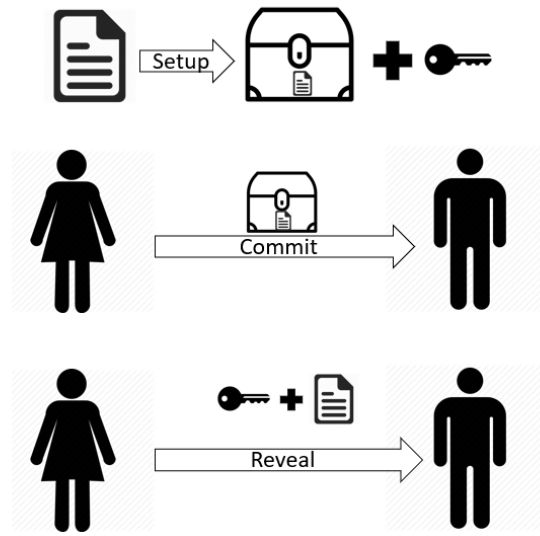

Fig. 1. Commitments

A. Protocol

The following steps are considered the basic protocol of commitment-schemes and vary only in their implementation. Often the protocol is shortened to only two steps, the commitment and the reveal (step 1 and 3), as these are the only steps requiring communication. This shortened version is also depicted in figure 1.

1) A **commits** to B
2) B keeps commitment, unable to read or process it
3) A **reveals** to B
4) B verifies the commitment

To elaborate this protocol, we use another common picture as shown in figure 1: putting a message in an unbreakable, non-transparent and locked box (shown in the *setup-step* of the figure).

For *step 1* of the protocol, Alice **commits** the box to Bob. Bob cannot see or alter the message inside the box. Picking up the example of the *coin-toss via telephone*, after *step 2* Bob would announce the outcome to Alice.

For *step 3* of the protocol, Alice **reveals** the commitment to Bob, sending him the key for the box (and commonly sending the message as well, proving that she is not only in possession of the key but also the secret itself).

If the key *fits*, Bob is able to unlock the box in *step 4* and compare the message provided by Alice with the message found in the box.

B. Attributes

The following attributes are required for commitment-schemes to be secure and successfully fulfill their purpose:

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

Except for viability, the fulfillment of each attribute will be shortly discussed in the regarding implementations.

For elaboration, the image of the box is fitting aswell:

Once locked inside the box, the message cannot be altered by Alice (or any other party). After she commits it, Alice is **bound** to the value, as there can never be a different value inside the box after this point.

Through the attributes *non-transparent and unbreakable* of the box, the **hiding**-attribute of the protocol is fulfilled: Bob cannot look into the box and does not know anything about the message if he does not have the key. The only way to properly gain knowledge about the message is the key provided by Alice.

The **viability** is given, as **only** the (correct) key will open the box. The correct key will always open the regarding boy, and no other key will *ever* be able to unlock the message.

There are two additional attributes based on the fact that we are working with computers:

1) Bobs are able compare commitments.
2) Commitments are *tradeable* and replicable - both for Alice and Bob, still keeping their primary attributes and are fully functional. This attribute is vital for the case study presented in section III.

The trading can be displayed with the box: If Bob_1 decides to give the box to Bob_2, Bob_2 is also unable to read or open it. Alice can now also contact Bob_2 with the key and the message, to which Bob_2 can open the box and compare the message, just like Bob_1 would do.

Also Alice is able to trade her key and message to any $Alice_2$, which enables $Alice_2$ to reveal her like the $real$ Alice would.

The ability to copy and compare commitments is not applicable to the example of the locked box. However, as most commitments rely on numbers, the *real* commitment consists of bits and therefore are easily duplicated.

For the comparison it's to add, that it's possible to have the same commitment for a different combination of messages and keys. However, as the key's should be chosen randomly, these collisions are highly unlikely (in case of the hash-based implementations as likely as hash-collisions themselves).

C. Additional Security-Measures

There are several *best-practices* which are not functional for the protocol itself, but are necessary to secure any party involved in the protocol and any application using the protocols. They will be shortly summarized and explained:

a) commitments should be one- (positive-) use only: This originates from the *reveal*-step, in which everything required to reveal the commitment successfully is transmitted. An eavesdropper would after the initial reveal be able to copy the required *credentials* and also reveal the commitment correct. To fix this issue, simply mark used commitments as deprecated (if they are further required), or delete them completely.

b) commitments should have a lifetime: (in time and/or tries) This behavior helps against brute-force attacks from exterior, given that the attacking party does not hold the commitment itself. If an aggressor has the commitment, he can start brute-force attacks locally (this holds true for eve and bob) which still requires **safe** cryptographic implementations. Additionally the lifetime (in e.g. days) is useful for Bob, as he has limited resources and should only keep *required* information.

c) traded commitments to a third party should be deprecated directly with first reveal: This is an extended version of the problem shown in paragraph *a)* of this subsection. Given there are multiple copies of the same commitment, and an eavesdropper knows the parties which hold a copy, Eve can successfully reveal the commitment to any party. For addressing this issue, the commitments need to be recursively deprecated throughout any party which the commitment was shared to. A common way to do this for Bob is to reveal the commitment by himself - this method does not require additional structures and also verifies that Bob knows the correct values.

However, if there is a larger number of parties involved, Eve can be *faster* reaching to the last Bob and reveal the commitment. Additionally there are many attacks that disturb the communication between Bob's, thus leaving more chances for Eve to reveal herself as Alice. Sharing commitments should be therefore only used when required.

d) messages must contain random parts: This rather trivial point is important for any implementation to fulfill any attribute connected to the *computational safeness* of hashfunctions and the discrete logarithm.

For every implementation based on commitment-schemes all of the above should be taken to account. There are several problems if only a single point is left out, including identity theft and server-malfunctions.

There are common libraries which support you in the goal of a secure implementation, e.g. an implementation in Haskell [HaHa] . The use of an open-source and **maintained** library is highly recommended.

II. Implementation

A. Hash-Based Commitments

For an easy introduction into the implementations, let's first introduce the hash-based commitments. These implementations rely solely on the cryptographic attributes of their regarding hash-function, making them relatively easy to understand, as every cryptocraphic process is *veiled* by a single function - leaving out complex mathematics.

It's therefore important, to choose a **cryptographic hash function** (todo: Source!). A *normal* hash-function does not fulfill the attributes as shown further below, and therefore cannot be used for commitments.

The basic protocol implemented with hash-functions:

1) Alice chooses a random value s
2) Alice produces $h = Hash(m \star s)$ and sends h and $Hash$ to Bob
3) Bob keeps $< Alice, h, Hash >$
4) Alice reveals herself by sending Bob m and s
5) Bob checks if $Hash(m \star s) \equiv h$

This implementation is parallel to the example with the box: The *key* to the box is the random salt, and the message is hidden behind it.

To choose a random salt value is necessary, as the domain of the messages is usually limited. Picking up the example of the coin-toss, Alice would be only able to commit to *Tail* or *Head*. Without a random value, Bob (and any Eve) could simply try both values and compare to the commitment.

While the example of coin-tossing is rather trivial, even bigger example such as *dates of birth* can be easily tried for multiple centuries.

Additionally, without the salt, using rainbow-tables and other dictionaries is possible.

It's also to mention that the message m should never be *really* valuable - as it's send in cleartext in *step 4* of the implementation.

The attributes required for a correct commitment-scheme are inherited directly by the attributes of a cryptographic hash-function:

Binding: After Alice created the hash, due to the **second-pre-image resistance** of the hash-function, she won't find any second message in feasible time that produces the same hash [1]. Therefore, she is bound to her value, as any other message would produce a different hash.

Hiding: After Bob recieved the hash, due to the **pre-image resistance** of the hash-function, the only way Bob can get to know m is by trying every possible value. As mentioned above, it's necessary to add a random salt for this attribute to be guaranteed.

[1] Bob needs to verify the used hashfunction. There are known hash-collisions to some hash-functions, which can be used to intentionally produce failures

To end this implementation, let's summarize the benefits:

- the implementation is easy to understand without further knowledge of mathematics
- iff the hash-function is cryptographic, the commitments are *safe*
- it's possible to commit *words* as messages, unlike other hash-functions, enabling *human-readable* examples

B. Pedersen Commitments

Instead of using hash-functions for their functionality, the pedersen-commitments gain their security from the un-feasability to extract roots from a finite body build by two (large) prime numbers. This assumption is the so called *discrete-log*-assumption.

Unlike the hashfunction, there is a bigger setup Bob needs to do before the main-protocol begins:

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

With these steps, Alice and Bob are sure to operate in the same finite body, which can be checked for *computational safeness*. The produces g is often called the $generator$, the v often $validator$.

Choosing the body can also be done by Alice, which would drastically benefit her, as she can purposely choose generators and validators which would enable her to construct non-unique commitments.

As a simplified example, Alice could choose the same g and v, making it possible for her to switch message and salt at will, while successfully revealing her commitment.

Even if Bob notices $g = v$, and rejects these kind of tricks, Alice is able to produce variables for certain collisions, which Bob can only notice with brute-forcing.

Therefore it's common for Bob to choose the body, as he is not about to commit values. If Alice notices problems with the safety of the given numbers, she can reject the communication.

The implementation of the pedersen-commitments:

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

The primary benefits of the pedersen-commitments are the following:

- the commitments always contain random parts
- the computational security can easily be increased by choosing bigger prime-numbers

C. Quadratic-Residues

Iff i need more content

III. Case-Study: Migrating user privileges in web-applications

. ToDo: Describe Web-Application, Requirements and the Solution

References

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first . . ."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

References

[HaHa] Pederson Commitment Schemes - a Haskell Library url: http://hackage.haskell.org/package/pedersen-commitment last seen: 29 Dezember 2018 maintained at the Massachusets Institute for Technology

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.