Two- and Multi-Party Protocols
JASS 05

Julian Traut

June 8, 2005

Abstract

After a short introduction we will present and discuss various protocols taken from [MOV96, GB01, Pfl96] and thereby introduce different approaches and techniques concerning cryptographic protocols. In the second part we will then outline a proof for the ability to construct secure protocols for any given functionality, as proposed in [Gol04].

Contents

Chapter 1

Protocols

1.1 Introduction

1.1.1 Why cryptographic protocols?

Talk about cryptography is mostly concerned with secure communication. Although this represents certainly the main part of cryptography, there is more to it than that. Since in the last decades dealing with things electronically has found its way into many areas of life, the need arouse for a way to deal with those new tasks from everyday life. This is where cryptographic protocols come into play.

1.1.2 What is a cryptographic protocol?

So what we need are algorithms that provide the ability to perform such 'real world' tasks, where two or more parties are involved, securly and efficiently. Basicly this is exactly the idea of an cryptographic protocol. In [Gol04] a more formal definition is given:

> [A cryptographic protocol is] a random process which maps m inputs to m outputs. The inputs to the process are to be thought of as local inputs of m parties, and the m outputs are their corresponding local outputs. The random process describes the desired functionality.

We will need this definition later, but for now it will suffice to stick to the rather informal one stated above. Although some of the tasks proposed on the following pages may appear of minor concern to the reader, the protocols deviced will be of didactical value in so far as they will introduce interesting approaches to the reader.

1.2 Devising a timestamping protocol in multiple steps

1.2.1 Task

As a first task we will consider timestamping services. Often it is crucial to certify that a document existed on a certain date. Normally one would make use of the services offered by a notary, but this requires a hardcopy of the document, which is to be physically taken to the notary and signed there. We want now to device a protocol for doing so electronically.
Alice wants to timestamp a document, so she can prove to Bob at a later time, that she created the document at this certain point in time. Bob and Alice both trust their friend Trent.

1.2.2 First Protocol

We will try an apporach analogous to the 'real world' procedure.

1. Alice sends a copy to Trent.
2. Trent stores the copy with the date and time he got it
3. Bob can now ask Trent for the document and the timestamp

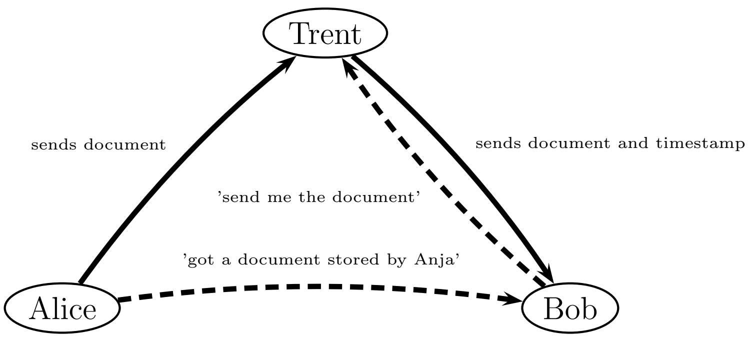

Figure 1.1: Timestamp 1

1.2.3 Discussion of First Protocol

There are several problems with this protocol. First there is no privacy, i.e. Trent has full access to the content of the document. Since Trent has to store the document somehow there may be additional problems with privacy, e.g. someone could gain unauthorized access to the database.
Secondly the protocol lacks efficiency. Trent has to store all documents he timestamps

for an indefinite time, this possibly resulting in a huge database.
Third there is the problem of errors that may occure during transmission and storage of the document. There is no way for Alice to ensure that the document has been transmitted and stored properly.
Last but not least we have the problem that it is necessary to invoke a trusted third party. This might not be possible or advisable. So we have to improve the protocol.

1.2.4 Second Protocol

For this protocol we make use of hashfunctions and digital signatures.
Instead of sending the whole document to Trent, Alice will do as follows.

1. Alice hashes the document and transmits the hash to Trent
2. Trent adds a timestamp to the hash, signs both and sends the result back to Alice
3. when Bob requests the document, Alice sends him the document and the signed and timestamped hash

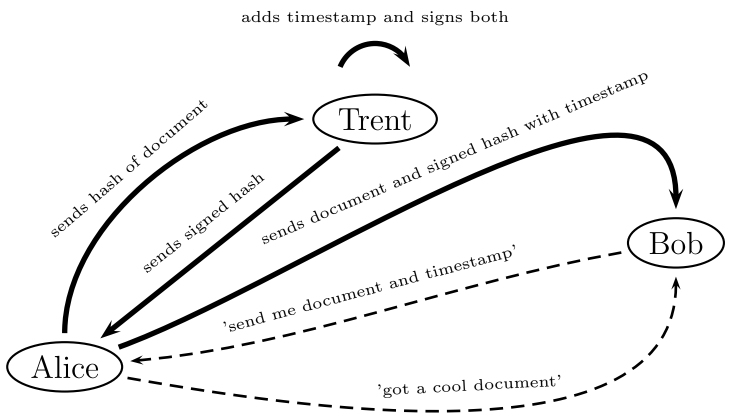
adds timestamp and signs both
Trent
sends hash of document
sends signed hash
sends document and signed hash with timestamp
Bob
'send me document and timestamp'
Alice
'got a cool document'

Figure 1.2: Timestamp 2

1.2.5 Discussion of Second Protocol

This protocol solves all problems but the issue of a trusted third party. Privacy is granted since Alice has to reveal only the hash of her document, from which no one can guess on the content. The problem of the huge database is solved since Trent does not need to store anything at all. Transmission errors are detected immedeatly since Alice can examine the signed hash right after execution of the protocol. The remaining problem is that Alice and Trent might collude. So what we need to do next is to emulate a trusted third party, so that we can do away with Trent.

1.2.6 Third Protocol

Instead of using Trent to timestamp the document, Alice will now invoke the protocol stated just before with several randomly choosen persons. When Bob wants to verify the validity of the timestamp, Alice can send him the document with the timestamp collection. Bob can then verify each timestamp in turn.

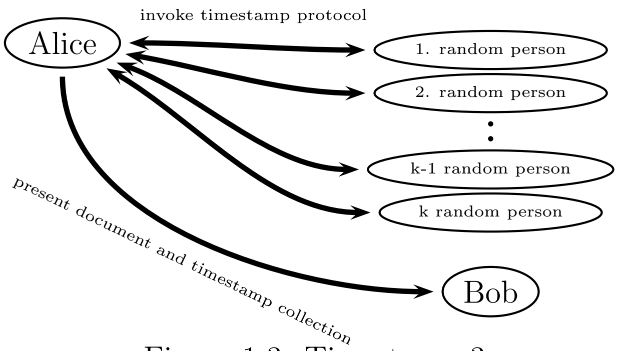

Figure 1.3: Timestamp 3

1.2.7 Discussion of Third Protocol

This protocol makes it very hard for Alice to cheat. The selection of the k random people could be done by using the hash of the document as input to some pseudo random number generator. When you choose k sufficiently high it should be impossible for Alice to collude with all k people.
There may occure a problem though, when someone who is selected by the random number generator to perform a timestamp is not able to do so, therefore a timestamp collection should be considered valid if some subset of those k people performed the protocol properly. In [MOV96] some further improvements are given, which involve the idea of linking a timestamp with previous timestamps generated.

1.2.8 Conclusion

We have now created an almost secure protocol for timestamping documents. This we have achieved by starting simply by transfering the 'real world' procedure into an electronic setting and then gradually eliminating problems. Thereby we have introduced a way to emulate a trusted third party by multiple untrustworthy ones. This was needed, since we wanted the third party actually perform something (timestamp the document), but often we only want to store a document for a certain period and then reveal the contend. We will deal with this issue in the next section.

1.3 Bit Commitment

1.3.1 Task

Just storing a document is normally no problem at all neither in 'real world' nor electronically. But if we impose an additional condition, namely that no one can alter the document after storage, the task becomes trickier. As before we could make use of a notary. For shortterm storage a sealed envelope would do, too. Imagine a magician guessing the card you will pick and writing down his prediction. But both won't work electronically.

1.3.2 Protocol

We will try to emulate the 'real world' procedure, just like we did, when devicing the protocol for timestamping. But we will in this case not emulate the notary but the idea of a sealed envelope instead.
We make use of a symmetric encryption algorithm like DES, for which Alice holds a secret key. The protocol is designed to commit to a single bit only, but this is easily enhanced to bit strings.

1. Bob will send a random bit string r to Alice
2. Alice will append her prediction bit and encrypt the result using her secret key
3. Alice sends the result to Bob
4. When Alice wants to reveal her prediction (e.g. after Bob has picked a card) she can simply send her key to Bob

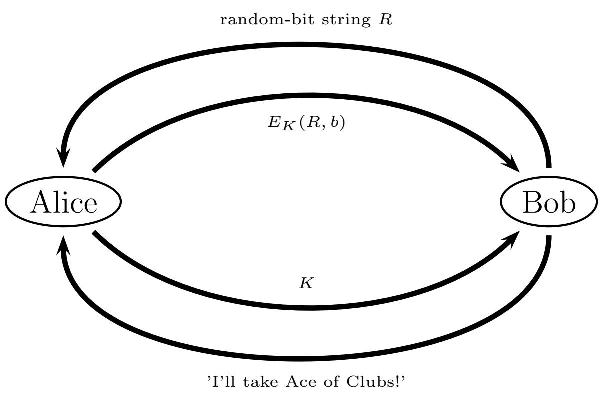

Figure 1.4: Bit Commitment

1.3.3 Discussion

The reader may have wondered about the necessity of the random bit string. This is essential to prevent Alice from cheating. If Alice had only to encrypt her prediction, she could prepare two keys in such a way that, for the given ciphertext, she could alter her prediction simply by presenting the other key. The random bit string makes this very hard. Alice would have to find two keys that invert the prediction bit but leave the random string untouched. A proper encryption algorithm makes this infeasible.
Obviously it is impossible for Bob to cheat, too, since he only gets ciphertext from Alice.
This protocol is used as a basic module for many more sophisticated protocols. Therefore there exist several other protocols for this task, e.g. involving oneway functions or pseudo random number generators.
Another basic protocol will be presented in the next section.

1.4 Coin Flipping

1.4.1 Task

Coin flipping may seem a rather peculiar task on first glance, but in several protocols it is necessary to agree upon a random bit sequence. This may be done by the following protocol. Obviously it is important that no party can predetermine the outcome of the coinflip, but on the other hand both parties should contribute to the result.

1.4.2 Protocol

The protocol makes use of a one-way function. It is in way very similar to the bit commitment protocol.

1. Bob chooses a random number, applies the one-way function on it and sends the result to Alice
2. Alice makes a guess on the random number(e.g. even/odd)
3. if Alice guessed right the result is heads, otherwise tails
4. Bob reveals the random number and thereby proclaims the result

1.4.3 Discussion

There are essentially two problems with this protocol. First the security rests only in the one-way function, but it is not formally proven yet that one-way functions exist at all. Second if Alice guesses on the least significant bit it is from utmost importance that it must be uncorrelated to the result of the one-way function, since otherwise Alice might influence the outcome of the coin toss. This is easily achieved by using a hard-core predicate as proposed in [Gol04]. As before there exists a bunch of other protocols for this task.

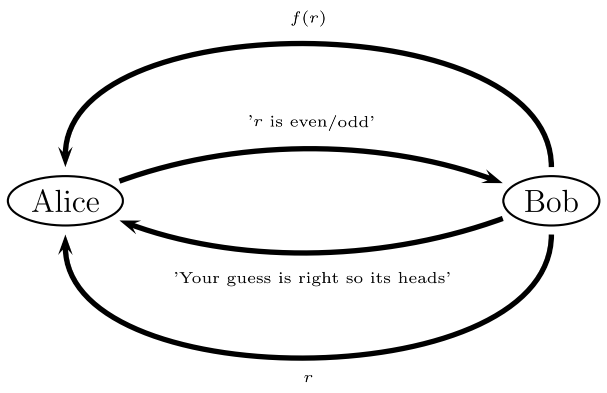

Figure 1.5: Coin Flipping

1.5 Oblivious Transfer

1.5.1 Task

Alice wants to send one message out of two to Bob. For Alice it should be impossible to know, which of the two messages Bob received and Bob should not be able to guess on the content of the other message. This task is really strange and seems to have no direct application at all. But it is important nonetheless, because it can be shown, as we will outlay in the second chapter that if you can perform this securely you can device protocols for any given functionality, that provide security, if not efficency.

1.5.2 Protocol

Alice posesses two public-key private-key pairs. Bob generates a random key for some symmetric algorithm.

1. Bob encrypts his secret key with one of Alice's two public keys, and sends the result to Alice
2. Alice decrypts this with both of her private keys (now she holds Bobs original key and another indistinguishable nonsense key)
3. Alice encrypts the messages with one key respectively and sends them to Bob
4. Bob decrypts both messages with his key. He will recover one plaintext and one random text

1.5.3 Discussion

There is no way for Alice to know which message Bob received, since she cannot distinguish between Bobs random key and the result she recovers when decrypting with the wrong private key. Bob cannot cheat either, because he cannot decipher the other message, since he does not posess the proper key. The only remaining problem is that

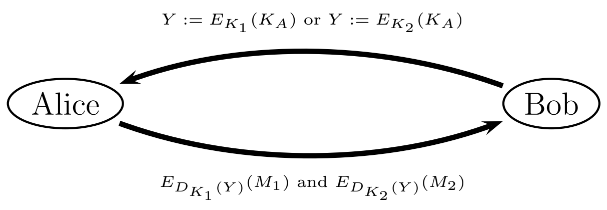
$Y := E_{K_1}(K_A)$ or $Y := E_{K_2}(K_A)$
Alice
Bob
$E_{D_{K_1}(Y)}(M_1)$ and $E_{D_{K_2}(Y)}(M_2)$

Figure 1.6: Oblivious Transfer

Alice could simply encrypt two identical messages. This may be prevented by revealing Alices key pairs at some later time, when secrecy of the other message is not required anymore.

1.6 Conclusion

In this chapter we have deviced protocols for various tasks. We have seen that there are several approaches to protocols, differing greatly from one another, e.g. transfering 'real world' procedures or emulating a trusted third party. We have used different means to achieve these tasks, e.g. one-way functions, public-key and symmetric cryptography, hash functions etc. What we have not done is to device multi-party protocols. For these please refer to the slides from the authors talk done on Jass 05 and the included exercises. The fact that there exist so many different solutions applying different techniques derived from all fields of cryptography makes protocols a very interesting subject of study. We will look now at protocols from a more theoretical angle to gain knowledge about the feasibility of devicing protocols in general.

Chapter 2

Proof Sketch - General Two-Party Computation

2.1 Preliminaries

2.1.1 Some Definitions

We will now look at following result taken from [Gol04].

> Assuming the existence of trapdoor permutations, one may provide secure protocols for *any* two-party computation (allowing abort) as well as for *any* multi-party computation with honest majority.

We will outlay the proof given in [Gol04], therefore we need some formal preliminaries. First we will get back to the formal definition of a protocol stated at the very beginning of Chapter 1.

> [A cryptographic protocol is] a random process which maps m inputs to m outputs. The inputs to the process are to be thought of as local inputs of m parties, and the m outputs are ther corresponding local outputs. The random process describes the desired functionality.

2.1.2 Ideal-Model vs. Real-Model

Since we want to prove that secure protocols exists, we should define security first. To do this Goldreich introduces two concepts: ideal-model and real-model
In the ideal-model the parties may employ a trusted thrid party, which is nonexistent in the real-model. We will consider the ideal-model secure, i.e. we will not consider actions that may not be prevented in ideal-model (e.g. refusing to participate), when defining security.

2.1.3 Adversaries

We will diffrentiate between a so called semi-honest adversary and a mailcious adversary. The semi-honest adversary follows the protocol correctly, but keeps track of all interme-

diate computations. This is not an unlikely situation in reality. While it might be hard to figure out every step of a complex application it is propably easy to read the memory resp. registers during execution.
The malicious adversary has more possiblities to interfere with the execution of the protocol. First he may simply refuse to participate, second he may substitute his local input and last but not least he may abort the execution of the protocol prematurely.

2.2 Protocols Used

We need two protocols during the proof.

2.2.1 Oblivious transfer

First we need a special version of the oblivious transfer protocol discussed in chapter 1. Instead of one message out of two we want to select one message out of four. The implementation of this protocol is analogous to the one presented before and thus left to the reader.

2.2.2 Computing $c_1 + c_2 = (a_1 + a_2) \cdot (b_1 + b_2)$

We will need a way to compute the result of a multiplication over $GF(2)$, where both parties hold shares to the input factors and after execution hold shares to the result. Both parties may not gather knowledge on either the input nor the output shares of the other party.

1. inputs P_1: (a_1, b_1); P_2: (a_2, b_2)
2. P_1 uniformly selects $c_1 \in \{0, 1\}$
3. Parties invoke oblivious transfer protocol from above (P_1 shall be the sender P_2 the receiver) with following inputs:
   P_1: $(c_1 + a_1 b_2, c_1 + a_1(b_1 + 1), c_1 + (a_1 + 1)b_1, c_1 + (a_1 + 1(b_1 + 1))$
   P_2: $1 + 2a_2 + b_2 \in \{1, 2, 3, 4\}$
4. P_1 outputs c_1; P_2 outputs result from OTP

It is easy to verify that this yields the desired result.

2.3 Proof for semi-honest adversary

Both protocols described above are secure with respect to a semi-honest adversary. Using them we can now create a secure protocol (with respect to a semi-honest adversary) for any given functionality as follows.

1. Break up the functionality into arithmetic circuits over $GF(2)$.

2. Create shares to own input wires for the other party, by adding a random bit which is sent to the other party
3. Now evaluate circuits one by one as follows
4. For multiplication circuits we use the protocol deviced above
5. For addition gates simply both parties add up their own input shares
6. Finally transmit the shares of other parties output wires and recover output

We now can device secure protocols in semi-honest mode for any given functionality. The protocols resulting are basicly not efficient but useful only as a theoretical model. What remains is a way to create protocols with respect to a malicious adversary. We will do this by forcing a malicious adversary to behave in a semi-honest way.

2.4 How to force semi-honest behavior

We will present as sketch only for this. There are mainly three tasks to fullfill. First we have to guarantee that the substitution of the local input only depends on the original input and not on the other parties input. We cannot prevent input substitution per se, but we can prevent it after the protocol has started. This is done during a so called input-commitment phase. We employ here the method of zero-knowledge-proofs. In many protocols it is necessary that a party selects some random bit. To enforce the true randomness of this bit, we set up a random-pad beforehands. This is done by a variation of the coin flipping protocol and zero-knowledge-proofs. Last but not least we have to force the adversary to comply with the protocol, i.e. to send only messages that result from its local input and the random-pad. Again we use zero-knowledge-proofs for this.

2.5 Conclusion

We have now found a way to create secure two-party protocols for any given functionality. This we have achieved by first starting from a semi-honest adversary and then transfer the resulting protocols to malicious-mode. We will not concern us with the multi-party case here, but the ideas are basicly the same as with two-party protocols. This result does not imply that there is no more work to do. While we theoretically can create secure protocols we have seen in chapter 1 that there are several approaches to this task and none of them employs the method proposed here since it is very unpractical. Since the triumph of electronic ways to deal with things over the conventional one has only just begun there will be a lot of interesting work in the future. Interesting topics not covered here but in the authors talk during JASS 05 are digital cash and electronic elections, which both will propably become important in the near future.

Bibliography

[MOV96] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[GB01] Shafi Goldwasser and Mihir Bellare. *Lecture notes on cryptography.* 2001.

[Pfl96] C. Pfleeger. *Security in computing.* Prentice Hall, 1996.

[Gol04] Oded Goldreich. *Foundations of cryptography: Basic applications.* Cambridge University Press, 2004.