An Approach for Security Protocol Design Based on Zero-Knowledge Primitives Composition

Očenášek Pavel

Faculty of Information Technology,
Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
ocenaspa@fit.vutbr.cz
http://www.fit.vutbr.cz/~ocenaspa/

Abstract. The paper deals with automated methods for the design of security protocols and their design using zero knowledge protocols, or protocols, where it is possible to use zero knowledge protocols such as subprotocols.. Specific emphasis is placed on the use of compositional method. The paper also include the example of protocol design.

Keywords: Zero knowledge, Security Protocol, Automation, Implementation.

1 Introduction

Zero knowledge protocols [4] [5] have been created because the traditional security protocols for leads to information disclosure in any running protocol, which if they accumulate sufficient quantities can be used for example for identity theft. When running Zero Knowledge (we ale use the abbreviation ZK) protocol does not log when running leak any information.

This paper presents the partial results of the research published within the thesis [1].

When running zero knowledge protocol, participant is trying to prove knowledge of certain information to the other party. For participants we use the abbreviation P (prover) for those who are trying to show that knows the information. We use the abbreviation V (verifier) for the participants which are trying to verify this knowledge. The main idea is the ability to carry proof of knowledge of information without learning anything other than that the evidence is true. This is not about proof in a mathematical sense, but it is a dynamic process between the two parties.

ZK protocols use for maintaining the secrecy such problems as factoring large prime numbers and the Knapsack problem. Compared to modern cryptography, ZK protocol are less time efficient, i.e. the run takes longer, which can be disadvantageous for RT applications, on the other hand ZK are not as computationally intensive and can be used such as embedded devices, Smart cards, etc.

The disadvantages of ZK are their usefulness, because it transmits no information at runtime, they cannot be used for transmission of messages or key distribution. But can be used for authenticating users and there are cases where it can also be used to

M.J. Smith, G. Salvendy (Eds.): Human Interface, Part I, HCII 2011, LNCS 6771, pp. 374–378, 2011.

confirm the validity of data entered, without the recipient party, which requires confirmation of the data he knew.

2 Zero Knowledge Protocol Design

Traditional notation of Feig-Fiat-Shamir [3] can generally be used for any other ZK protocol notation. ZK protocol will always have the stated sequence of messages: commitment, challenge, response, and final confirmation or rejection.

The main drawbacks of the proposal of separate ZK protocol thus lies rather in the creation of mathematical functions, which are possible to use in this type of protocols. This work would not be used as part of the proposal to address this ZK but rather deals with the question og how the method can be used for constructing more complex protocols, where ZK protocols are used as subprotocols for the identification of individual participants.

2.1 Input and Output Knowledges

The requirements for ZK protocol can be specified as follows:

- Input knowledge

A = {Ncommit, f(x,y)}
B = {Naccept, Nchallenge}

- output knowledge

A = {Nchallenge, Naccept}
B = {Ncommit, f(Ncommit, Nchallenge)}

Unfortunately, these requirements can not express order of messages, that B has to send *Naccept* after receiving *f(Ncommit, Nchallenge)* from the party A. We have to specify this information in requirements, or include the information that proper subject have to authenticate using ZK protocols.

3 Implementation

The program seeks to gradually meet the specified goals. As the primary goals, the selected objectives are defined in the protocol specification. The program seeks to gradually meet the objectives with the use of some primitive. If there is a primitive, which fulfills the goals, its assumptions become the part of new goals. Upon completion of all the assumptions, the primitive is added to the protocol and knowledge are added to the Prolog database, as defined in the primitive specification.

The program continues this way until all the goals are met. Primitives, from which the program selects appropriate, are defined in the program startup. All primitives are nondestructive, therefore it is not necessary to examine whether the addition of a primitive changes any assumptions on which it depended on some of the previous primitives.

There are other additional conditions placed for the primitives in the design process of the protocol. None of the constraints may not be in any state of the current protocol. If the primitive is found to satisfy a goal, but its implementation becomes the restrictive conditions, it is necessary to choose another primitive.

Primitives themselves must also logically follow. For this purpose, we use lists of preactive and postactive users. Preactive user of each primitive must also belong to the set of postactive users from previous primitive. In the case that is the first primitive of the protocol, the preactive user must be defined as an initiator of the communication in the protocol specification. If the conditions are not met, the protocol is not valid.

The program has no special interface and takes advantage of Prolog interpreter.

4 Protocol Example: Key Distribution

The presented example shows the protocol for distribution of the generated key between both participating subjects *a* and *b*, with the trusted server *s*.

```
actor(a).
actor(b).
actor(s).
server(s).
channel(a, b).
channel(b, a).
channel(a, s).
channel(s, a).
channel(b, s).
channel(s, b).
key(a, b, secret, 0).
key(a, s, secret, 0).
key(b, s, secret, 0).
knows(s, key(a, b, secret, 0)).
knows(s, key(a, s, secret, 0)).
knows(s, key(b, s, secret, 0)).
knows(a, key(a, s, secret, 0)).
knows(b, key(b, s, secret, 0)).
initiator(a).
```

```
goal(knows(a, key(a, b, secret, 0))).
goal(knows(b, key(a, b, secret, 0))).
forbidden(knows(s, x)).
forbidden(knows(spy, x)).
forbidden(knows(spy, key(a, b, secret, 0))).
forbidden(knows(spy, key(a, s, secret, 0))).
forbidden(knows(spy, key(b, s, secret, 0))).
```

The generated protocol is as follows:

```
1: a -> s: s, {|req(a, reqI(0, Kab0)), a, b, s|}Kas0,
           <req(a, reqI(0, Kab0)), a, b, s>Kas0
2: s -> a: a, <req(a, reqI(0, Kab0)), a, b, s>
3: a -> b: b, reqI(0, Kab0), a, b, s, <reqI(0, Kab0),
           a, b, s>
4: b -> a: a, <reqI(0, Kab0), a, b, s>
5: b -> s: s, {|req(b, reqI(0, Kab0)), a, b, s|}Kbs0,
           <req(b, reqI(0, Kab0)), a, b, s>Kbs0
6: s -> b: b, <req(b, reqI(0, Kab0)), a, b, s>
7: s -> a: a, {|Kab0|}Kas0, <req(a, reqI(0, Kab0)),
           Kab0, a, b, s>Kas0
8: s -> b: b, {|Kab0|}Kbs0, <req(b, reqI(0, Kab0)),
           Kab0, a, b, s>Kbs0
```

5 Conclusions

The presented approach implements the composition method presented in [2]. As we have showed in the example, the application is able to design new simple protocol upon the basic requirements and goals specification. The protocols can then include the zero knowledge primitives, as the basic composition primitives.

Acknowledgements. This study was supported by the Ministry of Education, Youth and Sports of the Czech Republic (MSM0021630528 and MSM0021630503) and projects FIT-S-11-1, FIT-S-11-2 and FEKT/FIT-S-11-2.

References

1. Safar, J.: Zero-knowledge protocol design, Master's thesis, FIT Brno University of Technology, Brno, Supervisor: Pavel Ocenasek (2010)
2. Choi, H.-J.: Security protocol design by composition. Technical report, University of Cambridge (2006)

3. Fiege, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: STOC 1987: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. s.210–s.217. ACM, New York (1987) ISBN 0-89791-221-7
4. Rosen, A., Goldreich, O.: Concurrent Zero-Knowledge. Springer, Heidelberg (2006) ISBN 3-540-32938-2
5. Simari, G.I.: A Primer on Zero Knowledge Protocols (2002)