Tatra Mt. Math. Publ. **29** (2004), 31–48

TATRA MOUNTAINS
Mathematical Publications

PRACTICAL DESIGN OF TWO-PARTY CRYPTOGRAPHIC PROTOCOLS

Pino Caballero-Gil — C. Hernández-Goya — C. Bruno-Castañeda

ABSTRACT. Two-party cryptographic protocols are here investigated as an important class of algorithms for which some insights into usual design are provided, and different issues related to it are discussed. So, the main objective of this work is to give a brief overview of such a field in order to propose various general schemes and to sketch several informal guidelines of a modular and generic methodology for improving the practice of robust protocol design. On the other hand, both one-sided and two-sided cryptographic primitives are here proposed as basic building-blocks for the systematic definition of more complex cryptographic protocols.

1. Introduction

A protocol is here understood as a set of rules or conventions defining an exchange of messages between a set of two or more parties. One of the major contributions of modern cryptography has been the development of advanced protocols created to provide secure services in distributed systems, and to enable several parties to electronically solve many real world problems. Several factors have combined to attract attention to the design and analysis of complex protocols. On the one hand, advances in cryptography have led to cryptosystems whose secure implementation requires the design of complex protocols. At the same time, applications have become more demanding due to the trend toward the general public availability of powerful computers and communications networks that generate new problems in protocols design.

A two-party cryptographic protocol may be defined as the specification of an agreed upon sequence of computations and communications performed by two entities in order to accomplish some mutually desirable goal, which is usually something more than simple secrecy. In fact, cryptographic protocols are always designed with a specific goal in mind. So, various solutions that allow to perform in the telecommunication world some usual actions as simple as flipping a coin

2000 Mathematics Subject Classification: 94A60.
Keywords: cryptographic protocol, secure two-party computation, attack, adversaries.

or putting a message in an envelope, and as complex as signing a contract or sending a certified mail are described in the form of different two-party cryptographic protocols. On the other hand, protocols designed to solve more unusual problems such as transferring information or identifying oneself, in both cases probabilistically, are also stated as two-party cryptographic protocols.

Several approaches to the design of cryptographic protocols have been carried out from different angles, some with the aim of developing a set of criteria that can be applied to cryptographic protocols in general, and others with the aim of developing criteria for a concrete protocol. The simplest approach to analyze systems that involve cryptographic protocols consists in considering them in an abstract environment where absolute physical and cryptographic security is assumed. The main disadvantage of this formal approach is that it does not address potential flaws in real implementations of concrete algorithms. On the other hand, the traditional approach in the bibliography has consisted in guaranteeing the security of real concrete protocols based on specific mathematical problems such as Quadratic Residuosity and Discrete Logarithm. This approach generally does not allow the composition of protocols in order to design more complex protocols, because it requires re-modeling the entire model and re-proving security. In this paper we propose a mixed approach where security conditions are guaranteed for certain types of real protocols that may be used as modules and composed in order to build complex protocols maintaining security conditions.

One of the main contributions of this work is the recognition that applications of cryptographic techniques by two parties in order to solve different problems may be productively viewed as an important type of algorithms (called two-party cryptographic protocols), which may be developed and explored following the mathematical models here proposed. Throughout this paper two-party cryptographic protocols are classified in two groups. The first of both is formed by one-sided protocols, whose representative here studied are Oblivious Transfer, Bit Commitment and Zero Knowledge Proof. The second group of two-party cryptographic protocols is composed of two-sided protocols such as Byzantine Agreement, String Verification and the Millionaires Problem. For every protocol, several simple design principles and general schemes have been deduced from the analysis of the best known algorithms by noticing some common features among them. Such general schemes have a large potential applicability, because they are independent of any logic or formal model and may be used directly as subroutines for general schemes of more complex protocols.

This work is organized as follows. Firstly, basic concepts and necessary tools are introduced in Sections 2 and 3. Afterwards, in Section 4, specific functional and computational notation used throughout the work is introduced. In Section 5, general properties corresponding to the definition of two-party cryptographic protocols are described according to the previously introduced notation.

In Section 6 special attention is paid to the general-purpose protocol of Oblivious Transfer and its different versions, whereas Sections 7 and 8 are devoted to the other one-sided primitives of Bit Commitment and Zero Knowledge Proof. On the other hand, Section 9 discusses several aspects of two-sided comparison protocols of Byzantine Agreement, String Verification and the Millionaires Problem. Finally, in Section 10, several conclusions are obtained and possible future works are mentioned.

2. Preliminaries

This work addresses the topic of secure distributed computing through the proposal of several general schemes for some two-party cryptographic protocols. In such a context, two parties who are mutually unreliable have to cooperate in order to reach a common goal in an unsecured distributed environment.

Typically, in specific cryptographic protocol design the computing power of one or both parties is supposed bounded, some unproven intractability assumption is required, and some kind of interaction is necessary between both parties. Also, in most two-party cryptographic protocols two general techniques are used, the so-called Cut-and-Choose and Challenge-Response methods. Cut-and-Choose technique consists in two stages, where firstly a party cut a secret information in several parts, and then the other participant chooses one of them. The goal of this technique is to achieve a fair partition of the mentioned information. The second method, Challenge-Response, is also formed by two steps. The first is a challenge from one of the parties to the other, and the second is formed by the appropriate answer.

On the other hand, cryptographic protocols are communication algorithms which usually employ cryptosystems. That's why their security depends both upon the strengths of the underlying cryptosystems and on the effectiveness of the protocols in the exploiting these strengths. Consequently, a poor and careless design may expose protocols to breaches in security, which can be the ideal starting point for various attacks without the need of cryptanalysis.

Roughly speaking, the design of cryptographic protocols include two basic phases corresponding to specification and verification. Up to now, most works have concentrated on this latter step, and systematic specification of protocols is almost an undiscovered area yet. In the sequel we give a brief overview of the most significant results in this field. Best-known formal methods to analyze cryptographic protocols that have been published may be classified into three types. Modal logic based approach is best represented by the BAN logic model for analyzing cryptographic protocols first published in [BAN90]. On the other hand, one of the earliest works that used the idea of developing expert systems to generate and investigate various scenarios in protocol design was [JMF87]. The third approach to apply formal methods to protocol verification was based on algebraic systems and first applied in [DY83]. Regarding research in the emerging area of formal and systematic specification of cryptographic protocols, a modular approach was proposed in [HT94], whereas a methodology to facilitate both specification and verification of protocols was presented in [GS95], and several basic informal design principles were proposed in [AN96].

3. Models

It deserves to be remarked that protocol design is extremely easy when the help of a Third Trusted Party (TTP) is possible. In such a case all input information may be given by both parties to it, and then the TTP can distribute corresponding outputs to each party. However, the enormous costs of extra communications, establishment and maintenance of TTP justify a search for secure and non-arbitrated protocols. In fact, the importance of cryptographic protocol design lies in the fact that TTP becomes non necessary. Typical solutions to avoid TTP in cryptographic protocol design include the use of two powerful tools: explicit complexity assumptions and random choices.

Two-party cryptographic protocols usually operate by series of message exchanges between both parties over a clearly defined communication network. So, the possibility always exists that one or both parties will cheat to gain some advantage or that some external agent will interfere with normal communications. The simplest situation occurs because each party may function asynchronously from the other party and make inferences by combining a priori knowledge with properties of the received messages, determining information that is not immediately apparent, so such inferences must be taken into account in determining security. In a worst case analysis of a protocol, one must assume that any party may try to subvert the protocol. So, when designing a two-party cryptographic protocol, one of two possible models should be considered.

On the one hand, the so-called semi-honest model is defined when it is assumed that both parties follow the protocol properly, but adversaries may keep a record of all the information received during the execution and use it to make a later attack. In order to prove the security in this model, the simulation paradigm is usually applied. According to this paradigm, given the input and output of any party, it is always possible to simulate through a probabilistic polynomial time algorithm his or her view of the protocol without knowing any other input or output. So, when the simulation paradigm holds, it is obvious that such a party does not learn anything from the execution of the protocol. On the other hand, the malicious model is considered when it is assumed that adversaries may deviate from the protocol.

It has been shown [GMW87] that any protocol being secure in the semi-honest model can be transformed into a protocol secure in the malicious model. This theoretical result has become an important design principle throughout the field of cryptographic protocols, because often it is easier to start from the design of a protocol that is secure in a semi-honest model, and then to transform it into a protocol secure in a malicious model, simply by forcing each party to prove that he or she behaves as a semi-honest party.

Consequently, the security of an interactive protocol should refer to its ability to withstand attacks by certain types of cheaters or enemies, in such a way that specific operating requirements like secrecy and reliability hold despite such possible attacks. On the other hand, since protocol design is usually based on the belief that certain particular computations are difficult, rigorous analysis of its security suffers the same limitation, and the best that can be hoped for is a demonstration that either the protocol is secure or that some explicit cryptographic assumption of the difficulty of a problem is wrong. So, careful analysis is required to certify the security of even relatively simple protocols.

4. Notation

We adopt notation that is common in the literature. Note that since bibliographic notation is not quite uniform, here we make compromises between uniformity of this paper and faithfulness to the original notation. So, throughout this paper the symbols A and B represent the parties Alice and Bob. The typical notation "$A \to B$: X" should be interpreted as "the protocol designer intended X to be originated by A and received by B", because the messages are assumed not being sent in a benign environment, so there is nothing in the environment to guarantee that messages are made by A or received solely by the B.

4.1 Functions

In order to formalize the notion of cryptographic protocols, f denotes a two-argument finite function, that is $f\colon X_A \times X_B \to Y_A \times Y_B$, where X_i and Y_i denote private input and output sets, respectively. On the other hand, s_i, r_i and f_i denote three private values corresponding to a finite value, a random choice and a function, respectively. The sub-indices $i \in \{A, B\}$ indicate both parties A and B, respectively.

Intuitively, a cryptographic protocol may be generally described through a function f whose public output is defined by the expression

$$\begin{aligned} f(x_A, x_B) &= f\big((s_A, r_A), (s_B, r_B)\big) \\ &= (y_A, y_B) = \Big(f_A\big((s_A, r_A), (s_B, r_B)\big),\ f_B\big((s_A, r_A), (s_B, r_B)\big)\Big)\,. \end{aligned}$$

In this way, at the end of the execution, each party receives the output of f_i. Observe that previous definition is independent of the sidedness view of secure two-party computation. In two-sided protocols, that can only be considered in a semi-honest model, both private outputs will be the same. On the other hand, one-sided protocols should be analyzed in both semi-honest and malicious models.

4.2 Machines

Following [GMR85], a two-party cryptographic protocol may be considered as a pair (A, B) where each party is described by means of an Interactive Turing Machine (ITM) with a read-only private tape and a write-only private tape. Both machines share a read-only input tape and a write-only output tape, and the protocol specifies one program for each machine such that both programs coordinate all the messages, and at the end each machine has in its own output tape, its own output message.

Briefly, on input (s_A, s_B), formed by private inputs of A and B, respectively, and random input (r_A, r_B) formed by private random tapes of A and B, respectively, protocol (A, B) executes a sequence of rounds, alternating between A-rounds and B-rounds. In an A-round (B-round) only A (B) is active and outputs a message that becomes an available input to B (A) in the next B-round (A-round).

Let E be an execution of protocol (A, B) on input (s_A, s_B) and random input (r_A, r_B). The public transcript of E consists of the sequence of messages exchanged by A and B, and is denoted by $t = \mathrm{TRANS}^{A,B}(s_A, r_A, s_B, r_B)$. On the other hand, the private view of A (B) consists of the triplet (s_A, r_A, t) ((s_B, r_B, t)), and is denoted by $\mathrm{VIEW}_A^{A,B}(s_A, r_A, s_B, r_B)$ ($\mathrm{VIEW}_B^{A,B}(s_A, r_A, s_B, r_B)$). Finally, the output of E consists of the strings (y_A, y_B) output by A and B in their last rounds of E, and denoted by $\mathrm{OUT}_A(s_A, r_A, t)$ and $\mathrm{OUT}_B(s_B, r_B, t)$ respectively.

5. Properties

An advisable methodology for practical design of cryptographic protocols may be described as follows. First the designer should have a clear idea of what the protocol should achieve in order to specify the goal, and computation and communication requirements that the protocol should satisfy, which implies the so-called correctness property. However, expressing the correctness criteria of a protocol is not a trivial task, because most protocols include the use of randomness and interactions, and are based on some difficult problem or cryptosystem. Also, since the difficulty of the problems and cryptosystems does not guarantee absolutely the security of the corresponding protocols, an essential task of their design should be the anticipation to any possible situation, which corresponds to the proof of fault tolerance (including protection of parties' secrecy). Finally, cryptographic protocols should be fair, which means that it should be clearly defined what every party gets through them.

According to previous notation, it is said that a protocol (A, B) securely computes a function f in a semi-honest model, and so that function f is securely computable, if the following conditions hold:

Correctness: Each party may obtain the correct output value of f on those input arguments that have been previously distributed between both parties, which means that

$$\forall s_A \in X_A,\ \forall s_B \in X_B,\ \forall r_A, r_B \text{ letting } v = \mathrm{VIEW}^{A,B}(s_A, r_A, s_B, r_B),$$
$$\mathrm{OUT}_A(v) = \mathrm{OUT}_B(v) = f(s_A, s_B).$$

A's Privacy: Any value that A could compute efficiently from certain output of f, could be computed directly from her private input and output, which implies

$$\forall s_A, s'_A \in X_A,\ \forall s_B \in X_B,\ \forall r_B \text{ if } f(s_A, s_B) = f(s'_A, s_B) \text{ then}$$
$$\mathrm{VIEW}_B^{A,B}(s_A, \cdot, s_B, r_B) = \mathrm{VIEW}_B^{A,B}(s'_A, \cdot, s_B, r_B).$$

B's Privacy: Any value that B could compute efficiently from certain output of f, could be computed directly from his private input and output, so

$$\forall s_A \in X_A,\ \forall s_B, s'_B \in X_B,\ \forall r_A \text{ if } f(s_A, s_B) = f(s_A, s'_B) \text{ then}$$
$$\mathrm{VIEW}_A^{A,B}(s_A, r_A, s_B, \cdot) = \mathrm{VIEW}_A^{A,B}(s_A, r_A, s'_B, \cdot).$$

Fault-tolerance: The security of the protocol should be stated under any kind of behavior from both parties or external viewers.

Fairness: Both parties should know the full description and possible outputs of f.

On the other hand, it is said that a function f is trivial if it is securely computable even if a cheating party has unlimited computational power.

Unfortunately, for most known cryptographic protocols no theorems regarding their correctness, fault-tolerance, privacy and fairness have been proved.

Instead, there are security reductions to prove that the protocols are secure as long as certain mathematical assumptions are true. Early work on this field concentrated on privacy as the main security criterion, but later it was realized to be inadequate, since many protocols provide services that are only indirectly related to privacy.

6. Oblivious transfer

Oblivious Transfer (OT) is a fundamental two-party protocol that is used to transfer a secret with uncertainty. It solves the following situation: party A knows a secret s_A that wants to transfer to party B in a probabilistic way such that the two next properties are fulfilled:

Meaningfulness: B gets the secret s_A with probability 1/2.

Obliviousness: B knows for sure whether he received the secret s_A, but A cannot determine whether the transfer was successful any better than random guessing.

OT protocol is one-sided, because party A acts only as sender and party B acts only as receiver. So, the functional definition of OT is as follows:

$$f_B\big((s_A, r_A), r_B\big) = s_A \qquad \text{if} \quad r_A = r_B \text{ (or } r_A \neq r_B)$$

According to previous definition, OT may be easily defined on a probabilistic cipher with probability 1/2. The idea of OT was first proposed by Rabin in [Rab81] where he described an algorithm based on the integer factorization problem. Two important applications of this protocol are electronic contract signing, random sequence generation in distributed environments, and certified electronic mail [BVV84].

It has been formally proved that Secure Two-Party Computation in general can be reduced to OT in the semi-honest model [IY87], [BG89] and that OT can be used as a primitive for the design of any two-party protocol [Kil88]. Furthermore, it has been established that the existence of one-way trapdoor permutations guarantees the existence of securely computable OT in the semi-honest model [Rab81].

In Figure 1 we sketch typical stages and characteristics of an OT based on the Challenge-Response and the Cut-and-Choose methods. In this general scheme A's secret is supposed to be a solution to a difficult problem, and some complexity assumption on the computer capacity of both parties is usually required. The first step implies the definition of a partition of a difficult instance of the original problem. In the second step it is required the use of a one-way function h that should have been previously agreed by both parties. Depending on the coincidence or difference between both secret random choices carried out in second and third steps, the transferred solution is a valid solution to the original difficult problem or not.

Set-up: $A \to B$ partition of an input problem instance $\{P_0, P_1\}$.
Challenge: $B \to A$ the output of a one-way function h on a random element from one of both sets $r_B \in P_j$, $j = 0$ or 1, $h(r_B)$.
Response: $A \to B$ the solution to the problem defined by her random choice of an element from one of both sets, $r_A \in P_i$, $i \in \{0,1\}$ and the information sent by B, $\mathrm{Sol}(r_A, h(r_B))$.
Verification: The secret solution is successfully transferred tp B depending on both participants' choices.

FIGURE 1. GS-OT algorithm

According to previous functional definition, $y_B = \mathrm{Sol}(r_A, h(r_B)) = s_A$ in case $r_A = r_B$ (or $r_A \neq r_B$). Correctness and privacy properties are satisfied by the general scheme GS-OT due to the following. B obtains a correct output of function f when both parties are honest. In case A attempts to transfer a non-existent secret solution, a TTP or a Zero Knowledge Proof might be used to guarantee correctness. Concerning privacy, after taking part in the protocol if B does not receive A's secret solution, then B cannot obtain it, since his polynomially bounded computing power does not allow him to solve the problem. Furthermore, A cannot guess B's secret choice, so she does not know whether B obtained the secret solution.

6.1 Variants of OT

The previous description of OT corresponds to its simplest version. There exist two especially interesting variations that are called one out of two OT (1-2OT) and one chosen out of two OT (1C-2OT) [EGL82]. The first one is used when A has two secrets and B wishes to obtain one of them without letting A know which one. So, 1-2OT may be functionally characterized as follows

$$f_B\Big(\big((s_{A1}, s_{A2}), r_A\big), r_B\Big) = s_{A1} \qquad \text{if} \quad r_A = r_B$$
$$\left(\text{and otherwise } f_B\Big(\big((s_{A1}, s_{A2}), r_A\big), r_B\Big) = s_{A2}\right).$$

The essential difference between a 1-2OT and a 1C-2OT is that in this latter case B is particularly interested in one of both secrets, so the corresponding functional definition in this case is $f_B\big((s_{A1}, s_{A2}), i\big) = s_{Ai}$.

A proof that the three versions of OT are equivalent can be found in [Cré87]. In these two last versions of OT, meaningfulness and obliviousness properties should be interpreted as follows:

Meaningfulness: B gets exactly one of the two secrets.

Obliviousness: B knows which secret he received but A cannot guess it.

GS-OT remains valid for both 1-2OT and 1C-2OT, but with some little modifications. In case 1-2 OT, the unique difference is in verification step where the solution that B receives, $\mathrm{Sol}(r_A, h(r_B))$, coincides with one of both valid solutions, depending on A's and/or B's random choices, r_A and r_B.

On the other hand, regarding 1C-2OT, there is a difference in response step where there is no A's random choice, so the solution sent to B is simply $\mathrm{Sol}(h(r_B))$ where r_B is randomly chosen by B in the set P_i indicated by his choice.

7. Bit commitment

Bit Commitment (BC) is a two-party cryptographic protocol that is used to simulate the two main characteristics of an envelope:

Unalterability: A cannot modify its content once she has sent it to B.

Unreadability: B can neither obtain the committed value inside the envelope nor any information about it until A opens it.

The first condition is equivalent to the mentioned correctness property and is known as *binding* property of BC. The second condition corresponds to the mentioned privacy property and is called *hiding* property in BC.

In BC functional description, the use of a trapdoor function whose inversion is only possible for A is required, $f_B(s_A, r_A) = h(s_A, r_A)$

According BC original definition where the committed secret is a single bit, it may be considered as a surjective mapping from a large domain to $\{0, 1\}$. So, a bit is considered committed by a random element in the pre-image of the mapping at an output value. From this point of view, BC may be considered a sort of a hash function. According to this mathematical formalization, binding property of BC implies that the corresponding mapping is a function. On the other hand, BC meets the hiding property if both distributions of elements in the pre-image of zero and elements in the pre-image of one are indistinguishable to B.

BC was first defined by Blum in 1982 [Blu82]. Since then, many interesting algorithms based on various typical cryptographic tools such as hash functions, secret keys ciphers, pseudorandom generators, discrete logarithms or quadratic residues have been proposed. Also, BC schemes have proved to be very useful as building blocks in the design of larger cryptographic protocols, so they may be considered the second main primitive of two-party cryptographic protocol design.

In most cases, B is supposed polynomially bounded and A knows a secret solution to a difficult problem that uses to commit to a bit s_A. A general scheme for BC based on the Cut-and-Choose technique is composed of the steps described in the Figure 2.

Set-up: $A \to B$ a partition of an input problem instance $\{P_0, P_1\}$.
Component: $B \to A$ the witness $h(s_A, r_A)$ obtained through a trapdoor function h on a random element $r_A \in P_{s_A}$, where $s_S = 0$ or 1.
Opening: $A \to B\, s_A$.
Verification: B checks the received information and discovers the secret.

FIGURE 2. GS-BC algorithm

Binding property is satisfied by the general scheme GS-BC, because if A modifies the commitment, then the fraud is detected by B in the verification step. On the other hand, the hiding property is guaranteed through the one-way transformation used in commitment step.

As we may deduce from both previous general schemes, there are many coincidences between GS-OT and GS-BC, but in this latter case, B's role is passive because this participant is limited to check the received information in the last verification step. So, BC may be considered a non interactive protocol since all the communications are one-way from A to B.

8. Zero-Knowledge Proof

A Zero-Knowledge Proof (ZKP) is an interactive two-party cryptographic protocol that allows an infinitely powerful prover Alice (A) to convince a probabilistic polynomial time verifier Bob (B) about the knowledge of some secret information without revealing anything about it [GMR85]. According to previous definition, ZKP has two possible results: to accept or to reject the proof, and the secret information could be a proof of a theorem, a factorization of a large integer, a password or anything verifiable, that is to say, such that there is an efficient procedure for checking its validity. ZKP have proven to be very useful both in Complexity Theory and in Cryptography. Concretely in this latter subject they have played a major role in the design of strong identification schemes [FFS88].

The functional definition of ZKP is as follows $f_B((s_A, r_A), r_B) = 0$ if B accepts the proof, (and otherwise $f_B((s_A, r_A), r_B) = 1$).

When a ZKP is designed for use in practice, an important question is the subject of computational bounds on the computing powers of the parties. So, in the original definition of ZKP, it is assumed that the prover has infinite computing power and the verifier is polynomial time bounded. On the other hand, the so-called Zero-Knowledge Argument is defined when A's computational power is limited to probabilistic polynomial time.

Three characteristic properties of ZKP are completeness (if the claim is valid, then A convinces B of it with very high probability), soundness (if the claim is not valid, then B is convinced of the contrary with very small probability), and zero-knowledge (B does not receive any other information except for the certainty that the claim is valid). This last property may be checked through the demonstration that the prover A can be replaced by an efficient (expected polynomial time) simulator, which generates an interaction indistinguishable from the real one. This property is usually proved through a constructive specification of the way the simulator proceeds. The main difficulty of this proof is to achieve that the simulator convinces the verifier about the knowledge of the secret information without actually having it. This problem is usually solved thanks to the rewinding capability of the simulator, which may use several tries to answer the verifier without letting him know how many tries the simulator has used.

Two variants of zero-knowledge may be distinguished depending on the assumed computing power of possible dishonest parties. Computational zero-knowledge arises when it would take more than polynomial time for a dishonest verifier to obtain some information about the secret, whereas perfect zero-knowledge involves that even an infinitely powerful cheating verifier could not extract any information. Both previous notions can also be characterized through the amount of computational resources necessary to distinguish between the interaction generated by the simulator and the verifier, and the one associated to the prover and the verifier.

The existence of computational zero-knowledge has been proven for any NP-problem under the assumption that a one-way function exists [GMW86], so it is natural that most known ZKP are computational ZKP. On the other hand, a demonstration that the existence of perfect zero-knowledge for an NP-complete problem would cause the Polynomial Time Hierarchy to collapse has been given [For87]. These important results imply that any time a message is sent, it may be accompanied with a ZKP of that the message is correct, which is applicable in general to protect distributed secure computation against malicious parties. On the other hand, generally the sequential composition of ZKP is again a ZKP, but, however, if we try doing the iterations in parallel, then the resulting protocol does not seem to be ZKP, but anyway that is an open question yet.

Generally, bit commitment, interactive challenge-response, and cut-and-choose techniques are basic ingredients of ZKP. So, A 'cuts' her secret solution in several parts, commits to them, and afterwards B chooses at random one of those parts as a challenge. Some of A's possible responses prove A's knowledge of the secret solution, whereas the others guarantee against A's possible fraud. Also typically ZKP consists of several iterations of the atomic subroutine described below, in such a way that by repeating it an enough number of times the verifier's confidence in the prover's honesty increases. Thus, the number m of iterations should be agreed by A and B according to their different interests.

Using all previously mentioned ideas, a general scheme based on five steps described in Figure 3 may be established to describe most known schemes.

Set-up: $A \to B$ a partition of an input problem instance $\{P_0, P_1\}$.
Iterations: The following steps are iterated for $i = 1, 2, \dots, m$:
- **Commitment:** $A \to B$ a witness associated to a solution of a random instance r_A, obtained through a one-way function h, $h(r_A)$.
- **Challenge:** $B \to A$ a random bit r_B.
- **Answer:** $A \to B$ the solution to the problem P_j, $j \in \{0, 1\}$, defined from both random choices r_A and r_B, and A secret s_A, $\mathrm{Sol}(r_A, r_B, s_A)$.
- **Verification:** B checks the received information.

FIGURE 3. GS-ZKP algorithm

In a ZKP defined according to this general scheme, correctness is guaranteed through completeness and soundness properties. Completeness guarantees correct execution of the protocol when parties act correctly, whereas soundness protects B against a dishonest party A who does not know the secret. On the other hand, privacy is reached through zero-knowledge, because this property assures that B does not receive any information on the secret thanks to his participation in the protocol.

9. Two-sided comparison protocols

Now we consider the problem of evaluating a concrete function by two parties on secret inputs. In a general description of a Two-Sided Comparison Protocol (TSCP) two parties A and B each having a secret (s_A and s_B) want that both learn the finite output of a comparison function $g(s_A, s_B)$, but none of them learns anything about the other party's secret. The main characteristic of this protocol is that it is a symmetric protocol because both parties do the same actions and obtain the same result. On the other hand, a one-sided version of this definition where only B obtains the output, would be more appropriate in the malicious model, because in the original one-sided version, a malicious party could stop the protocol once got the result. This general protocol has important applications on Electronic Voting, Mental Poker and Data Mining, and the main problem of its definition is the simultaneity of both parties' actions. It has been proved [Yao82] that all functions with finite domain and finite image can be evaluated through TSCP.

A general functional definition of TSCP is as follows:

$$f\big((s_A, r_A), (s_B, r_B)\big) = \big(g(s_A, s_B),\, g(s_A, s_B)\big)\,.$$

Figure 4 sketches typical stages and characteristics of a general TSCP for comparing binary strings, that is based on a 1C-2OT.

Set-up: Each party chooses at random $2n$ binary strings of length k, $\{r^A_{iO}, r^A_{i1}\}$, $\{r^B_{iO}, r^B_{i1}\}$, $i = 1, 2, \ldots,$

Transfer: Each party sends $1C{-}2\mathrm{OT}(r_{i0}, r_{i1})$ to the other party, who selects according to his/her secret binary string.

Computation: Each party computes the bit-wise addition of all the received strings, with the sum of his/her private strings defined by his/her secret string $\sum\limits_i r_{is_i}$ and sends the result to the other party.

Verification: If both additions are different, both parties deduce the difference between both secret strings. Otherwise, they do not know anything for sure.

FIGURE 4. GS-TSCP algorithm

Note that in the verification step both parties could deduce the equality of both secret strings in case both strings coincide, but this would be a probabilistic deduction, because the probability to fail in this deduction is 2^{-k}.

According to previous functional definition, if $s_A = s_B$, $g(s_A, s_B) = 0$, because $\sum_i 1C - 2\mathrm{OT}\big(r^A_{i0}, r^A_{i1}\big) + \sum_i r^B_{is^B_i} = \sum_i 1C - 2\mathrm{OT}\big(r^B_{i0}, r^B_{i1}\big) + \sum_i r^A_{is^A_i}$. Otherwise, $g(s_A, s_B) = 1$, which implies that no party receives any certain information regarding the comparison between both secret strings. Correctness and privacy properties are satisfied by the general scheme GS-TSCP due to the following. Both parties obtain a correct output of function f when they are honest, because if both secret strings coincide, both final sums also coincide. In case one of both parties attempts to do a non-valid 1C-2 OT, a TTP or a Zero Knowledge Proof might be used to guarantee correctness. Concerning privacy, after taking part in the protocol, both parties have received only random strings that do not allow them to deduce the other party secret string.

9.1 Variants of TSCP

Different concrete implementations of the general definition of TSCP are considered in the following sections.

9.2 Byzantine agreement

In the protocol described in [LSP82], and known as Byzantine Agreement (BA), both parties A and B each having a secret bit (s_A and s_B) want to agree on the same bit, which should be $s_A = s_B$ in case this equality holds. According to this definition, if one party receives a bit different from the one that he or she owns then he or she learns the other's bit, but if both parties receive the same bit that they own, then they do not learn anything about the other's bit. That is to say, if $s_A \neq s_B$, they learn it with probability 1/2, but if $s_A = s_B$ they do not learn anything.

In this case, the functional definition of BA is as follows:

$$f\big((s_A, r_A), (s_B, r_B)\big) = (s_A, s_B) \qquad \text{if} \quad s_A = s_B$$

are identical bits (and otherwise $f\big((s_A, r_A), (s_B, r_B)\big) = (r, r)$ where $r = g(r_A, r_B)$ is a random bit).

GS-TSCP remains valid for BA by limiting the length of binary strings to $k = 1$. In this way, if $s_A \neq s_B$ both sums coincide with probability 1/2.

9.3 String verification

String Verification (SV) is a sort of a generalization of a Byzantine Agreement to binary strings. In this protocol proposed in [RFW96], both parties A and B each having some secret n-bit string want to verify whether both strings are equal, but nothing more than that. There exists a general result proving that it is always possible to compute through STPC any function $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ using Boolean circuits.

In this case, the functional definition of SV is as follows:

$$f(s_A, s_B) = (0, 0) \qquad \text{if} \quad s_A = s_B$$

are identical strings (and otherwise $f(s_A, s_B) = (1, 1)$).

Again, GS-TSCP remains valid for SV by considering large enough values of length for the binary strings k, so that in the verification step both parties could deduce the equality of both secret strings in case both strings coincide with an almost null probability to fail 2^{-k}.

9.4 Millionaires problem

In the protocol proposed in [Yao82] and known as Millionaires Problem (MP), two parties A and B are supposed to be two millionaires who wish to know who is richer without revealing any other information about each party's worth.

The functional definition of MP protocol is as follows:

$$f(s_A, s_B) = (0, 0) \qquad \text{if} \quad s_A > s_B$$

(and otherwise $f(s_A, s_B) = (1, 1)$).

This third version of TSCP is different from the previous schemes in two important questions. First, it is defined on integer values instead of binary strings. Also, the comparison is not on equality or difference, but on greater or lesser value. Anyway, GS-TSCP may be easily adapted to be used for MP simply by considering the binary representation of both secret integers s_A and s_B, and by implementing GS-TSCP from the most to the less significant bits (left to right) to discover the most significant bit that is different between both secrets, because it determines the relationship. Note that according to this suggested implementation, a lower bound on the difference between both secrets is being transferred.

10. Conclusions

The main objective of this work has been to give some informal guidelines for the practical design of some of the best-known two-party cryptographic protocols. On the other hand, formal characterizations of definitions and general schemes for such primitives have been proposed in order to illustrate the feasibility and utility of applying mathematics to the exploration and design of cryptographic protocols.

This work has also emphasized several aspects regarding typical cryptographic protocol design such as: the existing relationship among different protocols, the important role played by certain cryptographic primitives as building blocks of more complex protocols, the presence of common schemes in distinct algorithms, and the use of typical ingredients such as interaction and randomness, and necessary complexity assumptions in the definition of most algorithms.

Regarding open questions, although it has been proved in [BOGG$^+$89] that the design of two-party cryptographic protocols is not possible without some complexity assumption, the study of possible hypothesis relaxations in some cases is being the object of our investigations at the moment. Also one of our main interests in order to guarantee practical applicability is to describe schemes based on minimum computing, communication and memory requirements. Finally, another open question that also deserves further study is a possible generalization of the presented methodology for designing multiparty protocols.

REFERENCES

[AN96] ABADI, M.—NEEDHAM, R.: *Prudent engineering practice for cryptographic protocols*, IEEE Transactions on Software Engineering **22** (1996), 6–15.

[BAN90] BURROWS, M.—ABADI, M.—NEEDHAM, R.: *A logic of authentication*, ACM Transactions on Computer Systems **1** (1990), 18–36.

[BG89] BEAVER, D.—GOLDWASSER, S.: *Multiparty computation with faulty majority*, in: CRYPTO '89, (Gilles Brassard, ed.), Lecture Notes in Comput. Sci., Vol. 435, Springer-Verlag, Berlin, 1990, pp. 589–590.

[Blu82] BLUM, M.: *Coin flipping by telephone: a protocol for solving impossible problems*, IEEE Computer Conference (1982), 133–137.

[BOGG+89] BEN-OR, M.—GOLDREICH, O.—GOLDWASSER, S.—HASTAD, J.—KILIAN, J.—MICALI, S.—ROGAWAY, P.: *Everything provable is provable in zero-knowledge*, in: CRYPTO '88, (Shafi Goldwasser, ed.), Lecture Notes in Comput. Sci., Vol. 403, Springer-Verlag, Berlin, 1989, pp. 37–56.

[BVV84] BLUM, M.—VAZIRANI, U. V.—VAZIRANI, V. V.: *Reducibility among protocols*, in: CRYPTO '83, (David Chaum, ed.) Plenum Publishing, New York, 1984, pp. 137–146.

[Cré87] CRÉPEAU, C.: *Equivalence between two flavours of oblivious transfers (cryptography)*, in: CRYPTO '87, (Carl Pomerance, ed.), Lecture Notes in Comput. Sci., Vol 293, Springer-Verlag, Berlin, 1987, pp. 350–354.

[DY83] DOLEV, D.—YAO, A.: *On the security of public key protocols*, IEEE Transactions on Information Theory **29** (1983), 198–208.

[EGL82] EVEN, S.—GOLDREICH, O.—LEMPEL, A.: *A randomized protocol for signing contracts (extended abstract)*, in: CRYPTO '82, (David Chaum, Ronald L. Rivest, and Alan T. Sherman, eds.) Plenum Publishing, New York, 1982, pp. 205–210.

[FFS88] FEIGE, U.—FIAT, A.—SHAMIR, A.: *Zero-knowledge proofs of identity*, J. Cryptology **1** (1988), 77–95.

[For87] FORTNOW, L.: *The complexity of perfect zero-knowledge*, in: Proceedings of the nineteenth annual ACM Symposium on Theory of Computing, 1987, pp. 204–209.

[GMR85] GOLDWASSER, S.—MICALI, S.—RACKOFF, C.: *The knowledge complexity of interactive proof-systems*, in: Proceedings of the seventeenth annual ACM Symposium on Theory of Computing, 1985, pp. 291–304.

[GMW86] GOLDREICH, O.—MICALI, S.—WIGDERSON, A.: *How to prove all* NP-*statements in zero-knowledge, and a methodology of cryptographic protocol design*, in: CRYPTO '86, (A. M. Odlyzko, ed.), Lecture Notes in Comput. Sci., Vol. 263, Springer-Verlag, Berlin, 1986, pp. 171–185.

[GMW87] GOLDREICH, O.—MICALI, S.—WIGDERSON, A.: *How to solve any protocol problem*, in: Proceedings of the nineteenth annual ACM Symposium on Theory of Computing, 1987, pp. 218–229.

[GS95] GONG, L.—SYVERSON, P.: *Fail-stop protocols: A new approach to designing secure protocols*, in: Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications, 1995, pp. 44–55.

[HT94] HEINTZE, N.—TYGAR, J.: *A model for secure protocols and their compositions*, in: Proceedings of the IEEE CS Symposium on Research in Security and Privacy, 1994, pp. 2–13.

[IY87] IMPAGLIAZZO, R.—YUNG, M.: *Direct minimum knowledge computations*, in: CRYPTO '87, (Carl Pomerance, ed.), Lecture Notes in Comput. Sci., Vol. 293, Springer-Verlag, Berlin, 1987, pp. 40–51.

[JMF87] CLARK, S.—MILLEN, J.—FREEDMAN, S.: *The interrogator: Protocol security analysis*, IEEE Transactions on Software Engineering **13** (1987), 274–288.

[Kil88] KILIAN, J.: *Founding cryptography on oblivious transfer*, in: Proceedings of the twentieth annual ACM Symposium on Theory of Computing, 1988, pp. 20–31.

[LLP82] SHOSTAK, R.—LAPORT, L.—PEASE, M.: *The byzantine generals problem*, ACM Trans. Program. Lang. Syst. **4** (1982), 382–401.

[Rab81] RABIN, M. O.: *How to exchange secrets by oblivious transfer*, Tech. Report TR-81, Harvard Aitken Computation Laboratory, 1981.

[RFW96] MONI NAO RON FAGIN—WINKLER, P.: *Comparing information without leaking it*, Comm. ACM **39** (1996), 77–85.

[Yao82a] YAO, A.: *Protocols for secure computations*, in: Proceedings of FOCS '82, 1982, pp. 160–164.

[Yao82b] YAO, A. C.: *Protocols for secure computations*, in: Proceedings 23rd Ann. Symp. On Foundations of Computer Science, 1982, pp. 160–164.

Received October 14, 2003

Department of Statistics
O. R. and Computing
Faculty of Mathematics
University of La Laguna
S–38271-La Laguna, Tenerife
SPAIN

E-mail: pcaballe@ull.es
mchgoya@ull.es
cbrucas@gobiernodecanarias.org