Chapter 15

Privacy with Cryptographic Protocols

In any organization there will always be one person
who knows what is going on.
This person must be fired.

CRYPTOGRAPHIC protocols can enforce privacy in distributed computation of functions (Goldwasser & Bellare 1996) and are a competitor of the techniques discussed so far. (Goldreich *et al.* 1987; Chaum *et al.* 1988b; 1988a; Ben-Or *et al.* 1988) show how cryptographic protocols can be compiled from protocols/functions for honest agents. For some combinations of concepts of security and types of attacks, cryptographic protocols obtained this way can be safe.

Example 15.17 *Without intractability assumptions, when at most a minority of the participants (<1/3) can make coalition, there exist cryptographic protocols that are safe even against active attacks (Chaum* et al. *1988a).*

Nevertheless, with most cryptographic protocols, if the assumptions do not hold, the attacker that does not belong to prevented types of attackers can discover everything easily.

Remark 15.1 *Typically, at the end of a computation, no agent can know whether anything of its privacy is saved.*

Many existing cryptographic protocols are targeted to the computation of functions based on addition and multiplication, but there exist other secure protocols for problems like elections and user identification. Backtracking uses extensively a "test" operator. Problems we encountered trying to produce a secure "test" operator are described in Annex B.

Hewlett-Packard, learning that a competitor for their main product, the laser printer, is the ink-jet printer, decided to also develop ink-jet printers in parallel. With the same principle in mind, in February'02, I started to research cryptographic protocols for discrete and numeric distributed satisfaction problems. The obtained protocols have some drawbacks such that a trade-off remains between the competing techniques. Cryptographic protocols and distributed algorithms will continue to coexist for a while, as laser printers and ink-jet printers.

15.1 Why DisCSPs and Secure Protocols?

You will surely ask:

1 *Why discussing cryptographic protocols in a thesis on asynchronous algorithms for DisCSPs?*

The question is reasonable since usually cryptographic protocols are simple synchronous translations of centralized techniques.

Actually an important motivation to work on asynchronous protocols is the belief that such protocols are useful for privacy in negotiations (Silaghi *et al.* 2000b; Meseguer & Jiménez 2000; Silaghi *et al.* 2001c). A first mention of properties of privacy in ABT appears in (Yokoo *et al.* 1998).

The first mention of a relation between solving DisCSPs and cryptographic protocols has been made by an author of (Choi *et al.* 2001) at CP'01.

Annex B describes a few research directions for cryptographic protocols that I have followed and that did not yet lead to satisfactory results. However, describing these negative results is important since they are not yet known among other researchers.

It is known that a respected personality blocked for many years the world's research of the now successful field of neural networks because personally he did not believe it could work. Similarly, by not knowing the different trade-offs of competing techniques for privacy, one risks to block prematurely one or the other still promising research directions.

15.2 Cryptographic Protocols for multi-party computation

Cryptographic protocols are fascinating through their original properties. Interactive proofs and zero knowledge protocols, are one type of such cryptographic protocols (Goldwasser *et al.* 1985; 1989; Babai 1985). The interaction proofs are more powerful than written proofs. The user can be asked to tell an *unexpected* half of a transformed proof and when he is able to answer, the chances that he is lying are reduced by 50%. Without revealing the proof, a series of such interactions reduces exponentially the chances of accepting a liar.

Secure multi-party computations are introduced in (Yao 1982). The main motivation behind them seems to be the fact that one system is easier to break than a set of systems. Imagine that a cryptographic key is stored on a computer system. Once an attacker breaks the system, the key is lost.

15.3 Secure multi-party computation

The idea in multi-party computations is to break the key into n pieces and to distribute these pieces to a set of n distinct systems (agents). The key cannot be reconstructed except if a subset of k agents, $k \leq n$, put their pieces together. The new version of the system is robust against the attackers that cannot corrupt at least k agents.

Two simple ways of splitting a number are:

- n points of a k degree polynomial that intersects the y-axis at an ordinate equal to the secret number (Shamir 1979).
- decomposition based on probabilistically transferred bits (Chor *et al.* 1985).

The second method allows for verifying the fact that the share of an agent has not been corrupted (modified) with less than O(log n) colluders. One of the first methods seems to be the one proposed in (Blakley 1979).

Once numbers are split and distributed, the idea (Yao 1982) is that distributed computations of an arbitrary agreed function can be performed over the distributed shares, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $k-1$). For (Shamir 1979)'s secret sharing, the computation of the sum function is efficiently implemented as a sum of the shares for corresponding samples. Multiplication is more complex and recent research has focused on improving its efficiency (Hirt *et al.* 2000).

Functions can be *compiled* unto secure cryptographic protocols. When a certain function has to be computed on shared secrets, all its steps can be performed using secure protocols for the corresponding steps. This can be done in such a way that any attacker that cannot find the initial secrets, cannot find anything else than the *official output* of the protocol (Goldwasser & Bellare 1996).

Setting	Adv. Type	t-privacy	Literature
cryptographic	passive	$t < n$	(Goldreich *et al.* 1987)
cryptographic	active	$t < n/2$	(Goldreich *et al.* 1987)
information-theoretic	passive	$t < n/2$	(Chaum *et al.* 1988b; 1988a; Ben-Or *et al.* 1988)
information-theoretic	active	$t < n/3$	(Chaum *et al.* 1988b; 1988a; Ben-Or *et al.* 1988)
i.-t. with broadcast	active	$t < n/2$	(Rabin & Ben-Or 1989; Beaver 1991)

Figure 15.1: Some available results on secure computations.

Techniques for compiling protocols for honest parties when less than n/2 players cheat, and that are based on intractability assumptions, are proposed in (Goldreich *et al.* 1986; 1987). (Chaum *et al.* 1988b; 1988a; Ben-Or *et al.* 1988) show compiling techniques for cases where less than n/3 players cheat and that are secure without intractability assumptions. (Ben-Or *et al.* 1988) details a quite efficient technique for computing polynoms with less than n/2 cheaters, when all participants comply with the protocol. A compilation of available results on multi-party computation by Thomas Duebendorfer is shown in Figure 15.1.

One can see cryptographic protocols as a way of replacing one trusted party with a community performing the same operations but in a secure manner.

15.4 Cryptographic definitions for Privacy

A clear distinction has to be made between cryptographic security and information-theoretic security (Chaum *et al.* 1988a; Ben-Or *et al.* 1988).

Definition 15.1 ((Chaum *et al.* 1988a)) *An input value, x_i, is* cryptographically secure *if gaining information about it, other than that obtained from the wished outcome, z, is thought to be computationally hard.*

The information-theoretic approach (non-Cryptographic) does not limit the computational power of the processors (Ben-Or *et al.* 1988):

Definition 15.2 ((Chaum *et al.* 1988a)) *An input value, x_i, is* unconditionally secure *if gaining information about x_i is impossible beyond that available from the wished outcome, z.*

Most multi-party secure protocols impose some kind of restrictions to the security. In the previously mentioned approaches, these restrictions are either computational-based, or a limit on the number of supported colluders, or both.

Some nice formal definitions of privacy that illustrate the restrictions on the number of supported colluders appear in (Ben-Or *et al.* 1988). When Q_i is the set of states and F the set of possible messages, $I_i : Q_i \rightarrow F$ is the input function for player i. (Ben-Or *et al.* 1988) denote $\prod_{i \in C} Q_i$ by Q_C. A vector $\langle a_0, a_1, ..., a_{n-1} \rangle$ is denoted by $\langle a_i \rangle$. By u_C we denote the sub-vector of u, $u = \langle u_0, u_1, ..., u_{n-1} \rangle$ that contains $u_i, i \in C$. Consider that protocols consist of rounds $j = 1, 2, ..., T$ and that A is an arbitrary set. $q_i^{(j)}$ denotes the state of player i after round j.

Definition 15.3 ((Ben-Or *et al.* 1988)) *A set C is* ignorant *in a protocol δ (for computing f), if for every set of initial states $\langle q_i^{(0)} \rangle$, every protocol δ_C that looks like δ and every function $g' : Q_C \rightarrow A$ there exists a function $g : Q_C \times F^{|C|} \rightarrow A$ satisfying*

$$g'(q_C^{(T)}) = g(q_C^{(0)}, f(\langle I_i(q_i^{(0)}) \rangle)_C).$$

Definition 15.4 ((Ben-Or *et al.* 1988)) *A protocol (for computing f) is* t-private *if every coalition C with $|C| \leq t$ is ignorant.*

The main results in the field are summarized in Figure 15.1, where t specifies the guaranteed t-privacy property.

Another property of secure protocols is their correctness as parametrized function on the possible Byzantine behavior of the participants. A coalition is *harmless* if any of its incorrect actions can only lead to their elimination from the problem.

Definition 15.5 ((Ben-Or *et al.* 1988)) *A protocol is* t-resilient *if every coalition C with* $|C| \leq t$ *is harmless.*

(Ben-Or *et al.* 1988) states that for every probabilistic function and every $t<n/3$ there exists a protocol that is both t-resilient and t-private while there are functions for which there is no $n/3$-resilient protocol.

15.5 Zero-knowledge

No presentation of cryptographic protocols is complete without the definition of zero-knowledge introduced in (Goldwasser *et al.* 1989). Intuitively, a proof made by a prover to a verifier in an interactive system is zero-knowledge if a polynomially-bounded malicious verifier cannot learn anything else than the existence of the proof.

An interactive system is composed of two agents, A and B, where A wants to prove to B the existence of a certain property $x \in L$, without letting B learn anything about x that it cannot compute alone in polynomial (in x) time and resources. An interactive proof consists of a series of rounds where A makes a statement, B tosses a random coin for choosing a challenge and A has to answer the challenge. Any failure of A to answer a challenge decides the failure of the proof. A sufficiently long sequence of successful answers by A exponentially increases the confidence of B in the fact that A knows a proof. The protocol ends successfully when B is satisfied with its confidence. Such a protocol is denoted (A,B).

Goldwasser describes the interactive Turing machine (ITM). An interactive protocol can be formalized as an ordered pair of ITMs.

Two notions are needed to define zero knowledge. The *indistinguishability of random variables* is meant to compare the language defined by the messages of the prover.

- Two families of random variables are perfectly indistinguishable when they are equal.
- Two families of random variables are statically indistinguishable on L when no judge differentiate any random polynomially-bounded (in $|x|$) samples in L.
- Two families of random variables are computationally indistinguishable on L when no polynomially-time (in $|x|$) judge differentiate any random polynomially-bounded (in $|x|$) samples.

The *perfectly/statistically/computationally approximability of a random variable U(x)* on a language L checks that there exists a probabilistic Turing machine running in polynomial time such that its acceptance result-function on its input x is perfectly/statistically/computationally indistinguishable from $U(x)$. This notion is meant to quantify the knowledge gained by a verifier.

Definition 15.6 ((Goldwasser *et al.* 1989)) *Let* $L \subset \{0,1\}*$ *be a language and (A,B) a protocol. Let B' be a verifier trying to cheat using an extra input H bounded polynomially. (A,B) is perfectly/statistically/computationally zero-knowledge on L for B' if the family of random variables defining the final knowledge of B' is perfectly/statistically/computationally approximable for the language L' obtained by concatenating any element of L with H.*

(A,B) is perfectly/statistically/computationally zero-knowledge on L if it is perfectly/statistically/computationally zero-knowledge on L for all probabilistic polynomial time ITM B'.

15.6 Multi-party computations and DisCSPs

A classical approach for applying a secure multi-party computation to a problem consists in choosing a trusted party solution and compiling its steps (Goldreich *et al.* 1987; Chaum *et al.* 1988a; Ben-Or *et al.* 1988).

There exist generic compiling techniques for functions composed of additions and multiplications. I will show now how DisCSP techniques can be transformed to such functions that can be easily compiled. The DisCSP techniques that I address is: *Finding a solution to a (Dis)CSP.*

15.6.1 Consistency Achievement

A related work (see Chapter 3) has been published by (Swain & Cooper 1988). (Swain & Cooper 1988) proposed an arc consistency approach based on a *connectionist approach.* A logic function modifies the activation property of the values of the problem. In its initial formulation, the detection of the termination is by detection of convergence.

Dynamic detection of convergence cannot be done securely, as this would reveal information concerning feasibility of certain tuples. A safe way of terminating the computation is by ensuring that the highest number of possible cycles was performed, namely nd. The name I give to a secure compilation of this technique is Secure Connectionist Arc Consistency (SCAC).

15.6.2 Secure Search of a First Solution

Imagine we want to solve a CSP $P = (X, C, D)$ where X is a set of variables $x_1, x_2, ...x_n$, C is a set of constraints and D a set of domains for X. The domain of x_i is D_i, whose values are $v^i_1, v^i_2, ..., v^i_{|D_i|}$.

Definition 15.7 (first solution) *The first solution of a CSP given a total order on its variables and a total order on its values is the first among solution tuples when these are ordered lexicographically.*

Let *gconsistent(P)* be a function:

$$gconsistent(P) = \begin{cases} 1 & \text{if } P \text{ has a solution} \\ 0 & \text{if } P \text{ is infeasible} \end{cases}$$

We will design now a set of functions: $f_1, f_2, ..., f_n$, $f_i : CSP \rightarrow \mathbb{N}$, such that each f_i will return the index of the value of x_i in the first solution, or 0 if no solution exists.

$$f_i(P) = \begin{cases} k & \text{if } P \text{ has the first solution for } x_i = v^i_k \\ 0 & \text{if } P \text{ has no solution} \end{cases}$$

Let us first design the functions $g_{i,1}, g_{i,2}, ..., g_{i,|D_i|}$. $g_{i,j} : CSP \rightarrow \{0, 1\}$.

$$g_{i,j}(P) = \begin{cases} 1 & \text{if } P \text{ has a first solution for } x_i = v^i_j \\ 0 & \text{if } P \text{ is infeasible for } x_i = v^i_j \end{cases}$$

$$g_{i,j}(P) = gconsistent(P \cup \{x_i = v^i_j\} \cup_{k<i} (x_k = v^k_{f_k(P)})) \tag{15.1}$$

$$f_j(P) = \sum_{i=1}^{|D_j|} i * (g_{j,i}(P) * \prod_{k<i} (1 - g_{j,k}(P))) \tag{15.2}$$

Lemma 15.1 *The functions g and f given by Equations 15.1 and 15.2 correspond to their definition.*

Proof. The properties can be checked recursively starting with g_1, k and f_1. □

It remains to find an efficient implementation of *gconsistent().*

Lemma 15.2 *The existence of a polynomial implementation for gconsistent() would prove that* $P = NP$

Proof. Satisfiability of a CSP is NP-complete. □

Unfortunately we have not yet proven that NP=P and for the moment we give a solution with exponential cost.

Remark 15.2 *It is improbable that one will ever find secure protocols more efficient than generate and test. This is due to the fact that any trimming of a branch reveals information when the "test" operator cannot be implemented securely (see Annex B).*

Let $SS(P)$ be the ordered set of all tuples in the cross-product of the domains of P. Each constraint c in the set of constraints C is a function, $c : SS(P) \rightarrow \{0, 1\}$. The secret parameters of the distributions are the various values $c(t)$ where t is a tuple. Let us define the function p, $p : SS(P) \rightarrow \{0, 1\}$, defined as $p(t) = \prod_{c \in C} c(t)$.

$$gconsistent(P) = \sum_{t_i \in SS(P)} (p(t_i) \prod_{k<i} (1 - p(t_k)))$$

Proposition 15.1 *Given the previous definitions of the functions p, gconsistent(), $g_{i,j}$, and f_i, and a (Dis)CSP P, the vector $\langle v^i_{f_i(P)} \rangle$ defines a solution of P (the first one).*

Proof. See the definition of the functions f. □

Remark 15.3 *The computation of the vector $\langle f_i(P) \rangle$ requires only additions and multiplications and can be easily compiled unto a secure protocol using any of the classic techniques mentioned in this chapter.*

The secret parameters of the computation are the values c(t).

Remark 15.4 *Actually whenever an element of the vector $\langle f_i(P) \rangle$ is 0, the computation can be stopped since P is infeasible.*

The secure algorithm obtained by compiling the computation of $\langle v^i_{f_i(P)} \rangle$ is referred to as Secure CSP Solver (SCSPS). To circumvent the exponential number of intermediary results and to have an acceptable space requirement, the computation of *gconsistent* should be done tuple after tuple.

15.6.3 Restricting security to arc consistency

As noticed in the previous subsection, the secure computation of a full solution can be very expensive. A compromise could have interesting properties. The alternative I think of is the use of secure arc consistency, SCAC, mentioned above, into sMDC (Silaghi *et al.* 2000g).

The obtained algorithm is called synchronous Maintaining Secure Connectionist Arc Consistency (sMSCAC). SCAC could be also used in asynchronous search, but the high cost expected for SCAC may make asynchronous versions less interesting.

15.7 Summary

In this chapter we first describe secure multi-party computations. We then show how classic secure protocols (for addition and multiplication) can be applied to CSPs and therefore to DisCSPs.