The Role of Cryptography in Database Security

Ueli Maurer
Department of Computer Science
ETH Zurich
CH-8092 Zurich, Switzerland
maurer@inf.ethz.ch

ABSTRACT

In traditional database security research, the database is usually assumed to be trustworthy. Under this assumption, the goal is to achieve security against external attacks (e.g. from hackers) and possibly also against users trying to obtain information beyond their privileges, for instance by some type of statistical inference. However, for many database applications such as health information systems there exist conflicting interests of the database owner and the users or organizations interacting with the database, and also between the users. Therefore the database cannot necessarily be assumed to be fully trusted.

In this extended abstract we address the problem of defining and achieving security in a context where the database is not fully trusted, i.e., when the users must be protected against a potentially malicious database. Moreover, we address the problem of the secure aggregation of databases owned by mutually mistrusting organizations, for example by competing companies.

1. INTRODUCTION

1.1 Scope of this Article

Classical database security (e.g. see [3]) relies on many different mechanisms and techniques, including access control, information flow control, operating system and network security, prevention of statistical inference, data and user authentication, encryption, time-stamping, digital signatures, and other cryptographic mechanisms and protocols.

It seems desirable to develop a systematic understanding of database security problems and their solutions and to come up with a framework. Ideally, such a framework should give some assurance that all relevant security problems have been addressed, and it can possibly point out new security issues not previously considered. It is a goal of this extended abstract and the corresponding talk to contribute to developing such a framework and identifying new research directions for fruitful collaborations of the database, the information security, and the cryptography research communities.[1]

A major aspect that requires closer examination, and which has partially been addressed in research directions like private information retrieval (PIR) [4], is to reduce the required level of trust into the database.

1.2 Limitations of Scope

The treatment in this paper is at a quite abstract level, without explaining how concrete techniques and protocols work. Some solutions proposed in this paper are of a theoretical nature and can become practical only when the computing power and communication bandwidth, or the techniques themselves, have been substantially improved. Moreover, the solutions and protocols are described in an idealized setting with synchronous communication channels. In a real-world setting such an assumption may not be fully justified, and the protocols addressing this issue are even more complex.

1.3 Outline

In Section 2 we briefly discuss information security from a very high-level perspective and compare the design of secure systems with the design of correct systems. We also distinguish between unilateral and multilateral security. In Sections 3 and 4 we discuss unilateral database security, first assuming the database, then the user to be trustworthy, where protection must be achieved against the other party's potential cheating. In Section 2 we sketch the bilateral security problem where both the database must be protected against malicious users, and vice versa. A general paradigm for building multilaterally secure systems, called secure multi-party computation, is reviewed in Section 6. Section 7 briefly discusses the secure aggregation of several databases as an application of secure multi-party computation.

2. INFORMATION SECURITY

One of the main paradigm shifts of the emerging information society is that information is becoming a crucial if not the most important resource. Information differs radically from other resources; for instance and it can be copied without cost, it can be erased without leaving traces. Protecting the new resource information is a major issue in the information economy.

[1]We use the term "database" in the most general sense, including more general information systems than simple traditional databases.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD 2004, June 13–18, 2004, Paris, France.
Copyright 2004 ACM 1-58113-859-8/04/06 ...$5.00.

Information security has proven to be a notoriously difficult topic. To understand and define security one must have a clear understanding of what a system is supposed to accomplish and in which ways a potential attacker can try to prevent the system from operating correctly. In this section we address these issues at a very general level. It serves as a basis to address database security in the later sections.

2.1 Unilateral Security

In many security-relevant applications, security is seen as a unilateral problem: Some system (or entity, or collection of entities) must be protected against a malicious outsider, often called an attacker. The system is secure if no attacker (with certain capabilities) can cause any (significant) deviation of the system from the specified behavior. This includes, for example, that the attacker cannot extract secret information.

In order to define security, one must hence define the system specification, i.e., what the system is supposed to do under normal circumstances, as well as the adversary's capabilities. Such a specification of capabilities can include the available computing power, access to side information, etc.

Typical examples of unilateral security problems are the protection of a computer system by security mechanisms of the operating system, as well as the protection of an organization's internal network against hackers, for instance by firewalls and intrusion detection technology.

Another, perhaps less obvious example of unilateral security is the protection of the communication between two parties against an eavesdropper, for example by encryption. Although both parties must be protected, the situation is nevertheless a unilateral one because the required protection of the two parties is (jointly) against an external eavesdropper, not against each other.

Database security is often seen as a unilateral security problem: The database system must be protected against outsiders and possibly also against potentially malicious users, but is itself assumed to be trustworthy.[2]

2.2 Multilateral Security

In contrast to unilateral security, many security-relevant applications require the protection of several parties, each against the potential misbehavior of some other parties, possibly against all other parties.

A simple example of bilateral security are on-line transactions where both the customer and the vendor want to be protected against malicious behavior by the other. In practice, such bilateral security issues are often not really addressed and instead "solved" by assuming that one of the parties (e.g. the vendor) is trustworthy.

Another such example, which needs some more explanation, is the classical software piracy problem. The software vendor has developed some useful functionality (e.g. a collection of statistical tools), while the customer wants to apply the functionality to his data. The (idealized) specification is that a customer who pays is allowed to use the functionality. This specification is implemented by the vendor by encoding the functionality in a software package, sending the software to the customer, and the customer running the software on his machine. However, this achieves more than the specification, in an undesirable way: The user cannot only *apply* the functionality to his data, he can pass this capability on to other parties (software piracy). This problem is sometimes addressed by extra hardware mechanisms, but usually it is addressed only by legal deterrence, which means that from a purely technical viewpoint (which we take in this paper) the user is assumed to be trusted.

There is a dual (for now quite theoretical) solution to this problem. Instead of having the vendor send the software to the user, the user could send his data to the vendor for processing. More precisely, the vendor could run a web-service which manages, stores, and processes a customer's data. In this case, the customer need not receive the software but must of course fully trust the service provider.

Ideally, one would like to solve the software piracy problem in a fair and symmetric manner such that neither the vendor must send the software to the user nor must the user send the data to the vendor. This is a typical example of a specification which could easily be solved if a mutually trusted party were available. This party could obtain the software and the user's data and perform all operations for the user, giving only him access to the result of the computation. We will discuss in Section 6 that many security solutions can be seen as the simulation of a virtual trusted party by the actual involved parties.

Another example of multi-lateral security, involving three entities, are on-line auctions. The auctioneer, the bidder, and the party offering an object need to be protected against possible fraud by another party (and, of course, also against external attackers). An even higher level of security is achieved if each party is protected against the other two parties cheating collectively with a joint strategy.

Yet another example of multilateral security is e-voting discussed below.

2.3 Defining Security

As mentioned above, in order to define security one must define the system specification, i.e., what the system is supposed to do under normal circumstances, as well as the adversary's capabilities. But how should one model the adversary if one wants to achieve security simultaneously for different groups of potential cheaters?

Let us address the system specification first. In the above software example, the specification is simply that the user should obtain the result of applying the software to his data. However, both discussed implementations achieve more, actually too much from a security point of view. The specification can be seen as an idealized system which performs exactly (and only) the desired operations.

As another example, a secure communication channel is an idealized system with three parties, the sender, the receiver, and the eavesdropper. The specification is such that the sender can choose a message, the receiver gets it, and the eavesdropper gets nothing. Encryption is a secure implementation of this specification if one can argue that the ciphertext seen by the eavesdropper gives no information (in a computational sense) about the message.

For e-voting the specification is that each voter can choose his vote and that all voters should learn the correct outcome of the vote evaluated according to the rules. The fact that votes need to be communicated is not part of the actual specification, but of course some form of communication is

[2]Note that one need not distinguish between users and outsiders. The outsider could be defined as a user with no privileges.

unavoidable. Like many other specifications, the e-voting specification can be easily implemented using a trusted party (the voting authority) who receives (securely) all votes from the voters, counts them, and announces the result. But it is doubtful whether in practice such a trusted system, secure beyond any doubt (also in case of a highly unexpected outcome), can be implemented. Secure multi-party computation discussed in Section 6 allows to simulate such a trusted party.

In a general setting with a set P of parties, the security requirements must specify for which sets $S \subseteq P$ of parties their collective cheating must be tolerated, meaning that for the remaining parties $(P \setminus S)$ the specification is still achieved. A security requirement involves a whole collection $\Delta = \{S_1, S_2, \ldots, S_k\}$ of such sets, which typically overlap. In case of unilateral security, where a system S must be protected, S is not contained in any of the S_i. In a sense, unilateral security can be seen as a setting where the union of the sets in Δ is strictly smaller than the complete set P of parties.

One can model such a security requirement by assuming a *central adversary* who can choose one set $S \in \Delta$ and corrupt the players in that set, where corruption means that the adversary takes full control of these players. Security means that for the remaining parties there is no essential difference whether or not the adversary is present. More precisely, for any set $S \in \Delta$ corrupted by the adversary, the security for the remaining parties is guaranteed. To make this more formal is beyond the scope of this extended abstract.

When there are many parties (say n), a typical setting is that one wants to tolerate the cheating of any t parties, for some $t < n$. In this case, $\Delta = \{S \subseteq P : |S| \leq t\}$.

2.4 Correctness vs. Unilateral Security

Let us briefly compare the two problems of constructing a *correct* system and constructing a (unilaterally) *secure* system to see that correctness and unilateral security are conceptually the same.

A system is a *correct* implementation of a specification if it behaves as specified. Correctness is relative to a specification defining the desired functionality. Ideally, a specification defines all interfaces to the system (i.e. all methods for interacting with the system) and the complete input-output behavior (for all possible parameter ranges and for all possible ways of interacting with the system.)

Similarly, a system is *secure* if it behaves as specified, even in presence of an adversary with certain well-defined capabilities. Like correctness, security is relative to a specification (which defines what security in the given context means.) Again, a specification defines all interfaces to the system (i.e., all methods for interacting with the system) and the complete input-output behavior. But in contrast to correctness, one also considers an adversary with a certain interface to the system, and the specification must be met for all admissible adversary strategies.

From such a high-level point of view, correctness and unilateral security are essentially the same. In both cases, the system must behave according to the specification, where in case of correctness one quantifies over all parameter ranges etc., and in the case of security one quantifies over all adversary strategies. This is why multilateral security is perhaps more fascinating (at least as a research topic) and more paradoxical than unilateral security.

3. UNILATERAL DATABASE SECURITY

As mentioned before, in a traditional model of database security, the database is (usually implicitly) assumed to be trustworthy, while outsiders and possibly also the users are considered to be potentially malicious.

It is unnecessary to give a precise specification of a database system. It suffices to consider a general system with a state space Σ and a set $\mathcal{Q}$ of operations (queries). Each query $q \in \mathcal{Q}$ is specified by a state update function $f_q : \Sigma \to \Sigma$ and an output function $g_q : \Sigma \to \mathcal{B}$, where $\mathcal{B}$ is the range of all possible replies a query can produce. There is a set $\mathcal{U}$ of users, and each user $u \in \mathcal{U}$ has certain privileges, i.e., is allowed to perform a certain subset $\mathcal{Q}_u \subseteq \mathcal{Q}$ of queries.[3] In this abstract view, also the privilege management subsystem and the logging subsystem are seen as components of the database. Any query that is logged hence also changes the state, even if it does not modify actual data.

We briefly discuss the most important security techniques relevant in a unilateral context. But we point out again that the unilateral database security problems can be seen as problems of the correct implementation of a specification rather than an actual security problem, although this view is quite unconventional.

3.1 User Authentication and Secure Communication

The most basic unilateral security problem is to establish a secure connection between any user and the database. The term "secure connection" captures both secrecy and authentication of the communication. Also user authentication can be viewed as being implied by the authentication of the connection. Of course, a concrete protocol for establishing a secure connection might involve subprotocols at different layers of the communication stack, and a user authentication step may be involved.[4] If one assumes a public-key infrastructure (PKI) to be in place, then establishing secure connections can be achieved by standard cryptographic mechanisms and protocols. We refer to [10, 12, 8] for a general discussion of cryptography.

3.2 System and Network Security

Like any information system, a database system must run on a clean operating system and trustworthy hardware, and it must be protected against attacks over the network. But both these issues, while affecting a database's security, are not database security issues and should probably not be considered as such.

3.3 Access Control

Access control is the most classical database security topic. The access control system is the database component that checks all database requests and grants or denies a user's request based on his or her privileges. (Here we assume that the user has been authenticated.)

Research in access control is concerned with developing policies and languages for specifying privileges, and with

[3]There are usually different categories of users, including system administrators, different types of internal users, and possibly a category of (unspecified) external users with restricted privileges.

[4]For example, one could establish an SSL connection (without client authentication) between client and server and then authenticate the user at the application level.

software components implementing a given policy. A subtle aspect of access control is that rights can be seen like any other data item. This also includes the right to grant rights, which is potentially recursive. In many commercial databases, however, access control is quite simple. Access control can also be content-based, meaning that the decision is based not only on which data records are requested, but also based on their content (e.g. based on keywords), or it can be based on the history of the user's previous requests.

3.4 Preventing Inference

The access control problem becomes even more subtle when the possibility of inference is taken into account, i.e., if one is concerned that a user might, from a set of legitimate queries, be able to infer further information he is not supposed to obtain. A well-known example is statistical inference, where several statistical queries can be combined to obtain information about individual entries, by carefully specifying the populations for the individual queries.

This problem has no clean solution since it is, ultimately, an artificial intelligence problem. If one sees access control as a database specification (not a security) problem, the inference problem illustrates that a complete specification of a database's functionality is highly involved and probably impossible.

4. UNILATERAL SECURITY FOR THE USER

4.1 The Problems

Let us first discuss a few examples to see what the users' concerns might be and why one might want to protect users from a malicious database.

EXAMPLE 1. Consider a patent database, for example owned by company A and open to the public. A competing company B would probably not want company A to learn which patents company B is searching, as this might leak information about company B's projects and strategy.

EXAMPLE 2. Consider a health information system storing personal data of users, which is accessible to various authorized parties (the user, doctors, hospitals, the user's health insurance company, the user's employer, etc.), each with specified and sufficiently restricted privileges. Users might have no choice but to join such a system. It is obvious that the users might be concerned about possible misuse, for instance when certain information is leaked to the insurance company or to his employer. Ideally, the user would like to hide the information from the organization running the database, but it seems that he has no choice but to trust the database.

EXAMPLE 3. In the above example, any of the entities accessing the database might be concerned about the correctness of the data received from it. It seems that the users have to trust the database that it answers queries correctly and that it maintains the database correctly.

4.2 Private Information Retrieval

Let us discuss the problem of hiding the users' queries from the database (see Example 1). Private information retrieval (PIR), proposed in several papers and formalized in [4], achieves this. This is at first paradoxical, but one trivial (though impractical) solution would be for the database owner to send the entire database to the user so that the user could evaluate the query himself (and neglect the rest of the data). This shows that PIR is possible in principle.

PIR takes place in a setting where the user is trusted, i.e., there is no information to be hidden from the user. The goal of PIR protocols is to reduce the necessary communication. We do not discuss the known results and protocols.

4.3 Computing with Encrypted Data

Let us now address the problem of keeping the data stored in the database (not the query) secret.

Encryption is the usual technique used to protect the confidentiality of data. If the users encrypt the information stored in a database in order to prevent the database from seeing the data, then the database queries are restricted to simple storage and retrieval operations, which is not very useful.

When the database is supposed to answer queries involving several encrypted fields, it seems that it must decrypt the data before evaluating the query. However, a technique called *computing with encrypted data* allows to solve this problem. The idea is that every data unit (e.g. every bit) is stored in encrypted form, where the key is not known to the database. The database performs the logical bit-operations specified by the query on the encrypted bits, thereby obtaining the encrypted result of the query, which it returns.

Current solutions to this require substantial communication for every logical gate to be evaluated, but if one could find a probabilistic bit-encryption scheme that is homomorphic with respect to a complete set of logical operations (e.g. the NAND gate, or the EXOR and the AND gates), then non-interactive computation with encrypted data would be possible.[5] This remains one of the most intriguing open problems in cryptography.

By using a universal circuit, which takes a description of a circuit as input, one could even hide both the query and the data from the database.

5. BILATERAL DATABASE SECURITY

In this section we consider bilateral security where both the database must be protected against malicious users and the users must be protected against a potentially malicious database.

This can be achieved by so-called secure two-party computation proposed originally in [13]. This technique allows two parties to compute any specification, including the specification of a user interacting with a database. A typical illustrative example is the so-called millionaires' problem: Two millionaires want to find out who is richer, without telling each other how wealthy they are. Using a trusted party, this task can easily been solved, but using cryptography one can solve this task even without a trusted party.

Unfortunately, it is impossible to achieve full security for both parties. Rather, one must assume that each party follows the protocol and only afterwards may try to find out more about the other parties inputs and data than provided by the protocol's output.[6] This is for example justified in

[5]This would also potentially allow to solve the software piracy problem: The user could send the data in encrypted form to the software vendor.

[6]Such restricted type of cheating is often called passive cheating or semi-honest, in contrast to full-fledged cheating which is called active cheating.

case of the millionaires' problem if the two gentlemen honestly and fairly perform a protocol such that neither of them can (or must) see the other party's input (i.e., wealth).

Full security against active cheating can be obtained by involving several parties, using a technique called secure multi-party computation.

6. SECURE MULTI-PARTY COMPUTATION

6.1 The Paradigm

Secure function evaluation, as introduced by Yao [13], allows a set $P = \{p_1, \ldots, p_n\}$ of n players to compute an arbitrary agreed function of their private inputs, even if some of the players deviate arbitrarily from the protocol. More generally, secure multi-party computation (MPC) allows the players to perform an arbitrary on-going computation with new inputs being given into the computation and new outputs being generated. This corresponds to the simulation of a trusted party [6, 7].

Security in MPC means that the players' inputs remain secret (except for what is revealed by the intended outputs of the computation) and that the results of the computation are guaranteed to be correct. More precisely, security is defined relative to an ideal-world specification involving a trusted party: anything the adversary can achieve in the real world (where the protocol is executed) he can also achieve in the ideal world [2, 11].

Many distributed cryptographic protocols can be seen as special cases of a secure MPC. For specific tasks like collective contract signing, on-line auctions, or voting, there exist very efficient protocols. Here we consider *general* secure MPC protocols, where general means that any given specification involving a trusted party can be computed securely without the trusted party. General MPC protocols tend to be less efficient than special-purpose protocols. We refer to [9] for a simple explanation of the MPC paradigm.

6.2 Specifying the Adversary's Capabilities

The potential misbehavior of some of the players is usually modeled by considering a central adversary with an overall cheating strategy who can corrupt some of the players. Two different notions of corruption, passive and active corruption, are usually considered. Passive corruption means that the adversary learns the entire internal information of the corrupted player, but the player continues to perform the protocol correctly. Active corruption means that the adversary can take full control of the corrupted player and can make him deviate arbitrarily from the protocol. If no active corruptions are considered, then the only security issue is the secrecy of the players' inputs.

One distinguishes between two types of security. Information-theoretic security means that even an adversary with unrestricted computing power cannot cheat or violate secrecy, while cryptographic security relies on an assumed restriction on the adversary's computing power and on certain unproven assumptions about the hardness of some computational problem, like factoring large integers.

6.3 Some Known Results

In the original papers solving the general secure MPC problem, the adversary is specified by a single corruption type (active or passive) and a threshold t on the tolerated number of corrupted players. Goldreich, Micali, and Wigderson [6] proved that, based on cryptographic intractability assumptions, general secure MPC is possible if and only if $t < n/2$ players are actively corrupted. The threshold for passive corruption is $t < n$. (The case $n = 2$ and $t = 1$ with passive security was already discussed in the context of the millionaires' problem.) In the information-theoretic model, where bilateral secure channels between every pair of players are assumed, Ben-Or, Goldwasser, and Wigderson [1] Chaum, Crépeau, and Damgård [5] proved that perfect security is possible if and only if $t < n/3$ for active corruption, and if and only if $t < n/2$ for passive corruption.

More generally, the adversary's corruption capability could be specified by a so-called adversary structure [7], i.e., a set of potentially corruptible subsets of players.

7. SECURE AGGREGATION OF DATABASES FROM MUTUALLY MISTRUSTING ENVIRONMENTS

Consider, as an example scenario, that it has been agreed that the National Statistical Office (NSO) of a country should publish detailed weekly (or even daily) statistics about the country's economic situation, involving detailed internal data of all companies. This requires the cooperation of the companies which must provide their data to the NSO. But this is in conflict with the companies' interest to keep such data confidential, at least until published in an official company report. If the NSO were fully trusted, this task could easily be solved in the obvious manner.

One can view the collection of all databases as an aggregated database to which only the NSO has some privileged access for statistical queries (and not more). This is the specification that should be implemented. In particular, the NSO should not learn any individual company data.

Secure multi-party computation allows to solve this problem, i.e., to implement this specification. Each database plays the role of a player in a secure MPC protocol. More generally, one can define arbitrary queries on such a virtually aggregated database, and they can be evaluated without any other information leaking from the individual databases.

Acknowledgement

I would like to thank Gerhard Weikum and the program committee for the invitation to speak at this SIGMOD conference, Martin Hirt, Yuval Ishai, and Renato Renner for very helpful discussions on database security and related issues, and Christian Konig for his support with preparing this manuscript.

8. REFERENCES

[1] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 1–10, 1988.

[2] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[3] S. Castano, M. Fugini, G. Martella, and P. Samarati, *Database Security*, Addison-Wesley, 1995.

[4] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. In *Proc. 36th IEEE Symp. on Foundations of Computer Science (FOCS)*, 1995.

[5] D. Chaum, C. Crépeau, and I. Damgård. Multi-party unconditionally secure protocols. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 11–19, 1988.

[6] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pp. 218–229, 1987.

[7] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multi-party computation. *Journal of Cryptology*, vol. 13, no. 1, pp. 31–60, 2000.

[8] U. Maurer. Cryptography 2000 ± 10. R. Wilhelm (Ed.), Lecture Notes in Computer Science, Springer-Verlag, vol. 2000, pp. 63–85, 2000.

[9] U. Maurer. Secure multi-party computation made simple. *Security in Communication Networks (SCN'02)*, G. Persiano (Ed.), Lecture Notes in Computer Science, Springer-Verlag, vol. 2576, pp. 14–28, 2003.

[10] A.J. Menezes, P.C. van Oorschot und S.A. Vanstone. *Handbook of Applied Cryptography*. Boca Raton: CRC Press, 1997.

[11] B. Pfitzmann, M. Schunter, and M. Waidner. Secure Reactive Systems. IBM Research Report RZ 3206, Feb. 14, 2000.

[12] B. Schneier. *Applied Cryptography*. Wiley, 2nd edition, 1996.

[13] A. C. Yao. Protocols for secure computations. *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 160–164. IEEE, 1982.