Server-Assisted Cryptography

Donald Beaver *
IBM/Transarc Corp.

Abstract.

Cryptographic tools to protect data and joint computations abound. But they tend to carry trust relationships to the extreme, relying on full trust in third parties, on heavyweight, "do-it-yourself" mechanisms, or on masses of equally-trusted peers (as in secret sharing and secret computation). Trusted parties provide simple, elegant and efficient solutions but necessitate concentrated risk; threshold computations are prohibitively expensive but enjoy greater robustness.

This work investigates tools to change and accommodate trust relationships in a more flexible and gradual fashion, replacing discrete trade-offs between risk and complexity by a continuum of options. In particular, it proposes a new architecture for cryptographic tools, called *server-assisted cryptography*, in which lightweight clients obtain transferable and composable cryptographic resources from one or more third-party service providers. In contrast to TTP architectures, however, information flows in one direction only – from service provider to client – greatly reducing the trust placed in third parties.

* Currently at: CertCo, 55 Broad St., Ste. 22, New York, NY 10004; beaverd@certco.com.

1998 NSPW 9/98 Charlottsville, VA, USA

1 Introduction

The natural evolution of large systems follows several common patterns, including division of labor, specialization, compartmentalization, decentralization, and differentiation. As overall functionality increases, simpler and specialized tasks are identified and offloaded to specialists.

To some extent, the design of security architectures follows these evolutionarily-proven tendencies. Kerberos [SNS88], for example, assigns responsibilities for managing principal and password information to specialized security servers, so that other components (*i.e.* clients and generic servers) do not have to maintain, secure, and coordinate individual databases. But security seems to face obstacles to a continued evolution, particularly in that increasing the number of "specialists" paradoxically decreases security by enlarging the vulnerable trusted computing base.

In fact, the idea of using a trusted computing base is itself a limiting factor, since it implies a very simplistic trust model: one set of components is completely trusted, and the rest are not. While this may suitable for a world of isolated, fortress-like domains, it is hardly rich enough for large-scale, decentralized, societal systems.

The cryptographic tools on which much of information security relies are not much help. They, too, rely on oversimplified trust models: trust yourself and nobody else; trust yourself and a trusted third party (TTP); or trust society (*i.e.* trust that a majority of components are reliable). These trust relationships have one thing in common: trust is all-or-nothing.

Just as granting *least privilege* to users has been identified as a desirable characteristic for authorization, we look to *least reliance* on third parties as a compelling design principle for decentralized systems. When reliance is limited to certain properties or behaviors, there is no need to describe trust relationships in the crudest, all-or-nothing

fashion.

The obvious problem is whether it is possible to produce feasible security architectures with richer (and more complicated) trust models. This work takes a look at ways in which specialists can assist in increasing security and robustness without having to be absolutely trusted. Because existing cryptographic tools do not suffice, we also describe new cryptographic techniques to support this enrichment of trust models in information security.

1.1 Systems Evolution

To motivate our demands for different infrastructure and richer trust relationships, we first take a quick, *ad hoc* look at the evolution of large systems, both passively (as in nature) and artificially (guided by human design). Certain patterns reveal themselves time and again:

• **Division of labor.** By assigning a class of tasks to a particular individual, the individual is freed of the costs of context switching, and task management and completion is simplified.

• **Specialization.** As system functionality increases, all-purpose components become too large and complex to maintain, let alone design and analyze. It is easier to analyze and control the behavior of the overall system when the behaviors of components are restricted and simplified. Improvements in the performance of specialized tasks are easier to implement when they affect only a small fraction of components, namely those responsible for the given tasks. When any general-purpose component can be called to perform a critical task, every component must be equally protected; in contrast, specialization permits efforts to maintain integrity and assurance to be focused on critical components.

• **Replication.** Specialization can increase instability insofar as specialists become more critical. The simplest way to avoid risky dependence on unique or overloaded specialized components is to replicate them. Of course, when the compromise of a component can lead not just to loss of its functionality but to compromise of other properties (such as privacy), replication can increase risk. Note that certain cryptographic tools such as risk dissipation can enable replication without incurring this increased threat of compromise – at least, theoretically speaking.

• **Compartmentalization.** The reach of error or maliciousness can be confined by dividing a large system into autonomous domains. Scalability is difficult without the simplifications provided by compartmentalizing system management.

• **Differentiation.** Robustness can be enhanced by diversity, both in implementation of given services and in the way those services are divided and assigned. Regardless of its benefits, differentiation often occurs as an inevitable result of enormous scale and local autonomy.

• **Increased functionality.** As efficiency and stability increase, there is greater room to expand the functionality provided by and within a system. Thus complexity continues to grow, leading to further division of labor and specialization.

• **Translation.** Interoperability requires that the functionality provided by one domain be mapped into that requested by another. The larger the scale, and the looser the coupling across the entire system, the greater the need for translation and "glue."

In artificial systems in particular, there are the additional constraints of simplicity and scalability. These put greater pressure on division of labor and tend to increase the number of critical components. In order to scale reliably, however, blind reliance on such components must be lessened.

1.2 Security Architecture: An Exception?

Clearly, many systems architectures have applied the patterns listed above to security itself, dividing the labor of specialized authentication and principal management and assigning the lion's share to centralized (within a local system) security servers. Firewalls take on the specialized responsibility of filtering unwanted access to local domains. Information security does benefit from established patterns in limited ways, but are current security designs expandable to scales exceeding single enterprises and institutions?

1.2.1 Trusted Computing Base

Most system security architectures to date have indeed divided the labor of secure operations. In a classical, centralized system, the "trusted computing base" contains the specialized components who carry the responsibility of enforcing security. This

model has been the approach of choice for situations in which fortress-like protection is suitable.

In a more loosely coupled and decentralized environment, there may be "trusted third parties" (who may or may not be "members" – *i.e.* under the autonomous control – of the given system). Kerberos authentication servers are a common example. Certification authorities are another; they may or may not be part of the local system. Yet another example is a key escrow agent, who holds copies of private keys or shares of them, but is not under the control of or related to the parties on whose behalf it holds the keys.

TTP's are the natural extension of the trusted computing base approach to the networked world of distributed systems.[1] For reasons we discuss presently, TTP's require a perhaps unnecessarily strong trust model for *decentralized* systems.

1.2.2 From Enterprise to Internet

As interactions and transactions move from the domain of large but finite enterprises to decentralized and far-reaching internets, the restrictions and disadvantages of TTP and multiparty-based architectures are exacerbated. It is more reasonable for an enterprise to rely on a TTP when that TTP is under the autonomous control and management of the enterprise. When IBM and Microsoft engage in a transaction, however, whose TTP should they trust? When a Finnish student buys an encryption package from an American site, should she rely on the discretion of a US-government approved third-party transaction server?

The increase in *decentralized* distributed computations over *autonomously managed* distributed computations has a deep impact on deciding how security architectures should be designed. Trusted third parties can and will play an important role in the evolution of these architectures, but the increasing lack of common management and control makes TTP solutions more complicated.

Several aspects of societal interaction apply to enriching security and trust management in loosely-coupled environments. First, there is an increased need to be able to choose TTP's flexibly, from a pool of neutral (and properly motivated) third parties. (This will both complicate and simplify trust management, since a greater number of reputations or judgements may need to be accommodated, while a random selection of assistants from a yellow-page directory may lessen the estimated risk and the need for precise risk assessment.) Second, it may be useful to avoid placing all the eggs in the basket of a single TTP; but this safety motivation should avoid increasing demands on TTP cooperation and interoperability, which may be hard to impose. And third, it is important to restrict the amount of information flowing to the TTP's, because they do not necessarily share the same interests as any of the clients.

[1] We contrast "distributed," which connotes having common design and/or supervision, with "decentralized," which applies to diverse and loosely-coupled systems.

1.3 Trust Models in Cryptography

To see how cryptographic tools might help support new information security designs, we must first consider what cryptography already provides. As mentioned earlier, cryptographic tools tend to follow one of three, all-or-nothing trust models: trust oneself; trust a server; or trust the group.

• **Self-protection.** Encryption and key exchange are canonical examples of do-it-yourself protection. Moderate to heavyweight computations must be performed by the individual. A private, high-quality source of randomness is necessary. Rather than trust anyone to supply such random bits and keep them discreetly hidden, an individual generally trusts only himself to produce some moderate amount of randomness. Where this randomness is limited, expensive pseudorandom expansion is required.

In abstract cryptographic protocols where the behavior of another party is in question, zero-knowledge proofs allow each individual to check the veracity of certain facts, usually those that attest that the other party has followed prescribed protocol steps. The verifier trusts only himself.

• **Trusted third party.** Digital signatures and certificates require trust in the mapping from signing key to identity. Unless the individual who relies on signed data is able to verify that mapping directly (self-trust), he relies on Certificate Authorities, namely TTP's, to provide it. The mapping is either completely trusted, or ignored.

Many protocols assume synchronization or sequential ordering of messages. Accessing the correct time is usually assumed possible by fiat, or explained away with the availability of beneficent and secure system clock, *i.e.* a TTP.

• **Dissipating Risk.** Cryptographers (not exclusively!) have long recognized that placing trust in single components is a dangerous habit. Their primary response has been to develop a set of tools for dissipating risk among multiple individuals, with the guiding principle that while any given individual may be corrupt, the majority are probably reliable.[2]

The principal tool for implementing such democratic methods is *secret sharing* [Bla79, Sha79], which allows sensitive information to be split among several components in such a way that a majority of shares is needed to obtain any information whatsoever about the secret.

These methods have been extended to enable the computation of some function of already-shared secrets (such as their sum) without revealing those inputs [GMW87, BGW88, CCD88]. Theoretically speaking, groups of components can carry out general-purpose computation (such as ticket creation) without localizing the sensitive, intermediate data at any point.

Apart from the relative complexity of these multiparty protocols, this approach exacerbates certain disadvantages of TTP-based architectures. Bottlenecks are increased: each original TTP operation now involves communication among several servers. Physical security becomes more complicated: rather than a single strongbox, several are needed and in different locations – even though the dissipation of risk reduces the impact of compromising individual strongboxes. Requests for security services are more complicated, requiring either a gateway (which then becomes a TTP!) or direct communication with all the servers.

1.4 Modifying Trust Relationships

While a boolean, Trusted Computing Base approach is reasonable for a strongly confined and simple system, a richer trust model is needed to support increasing numbers of critical components and services and the unique sorts of trust relationships that occur among them.

Cryptography has left a gap in providing support, since it either assumes that certain components will be absolutely trusted, or that all individuals will wrap each interaction with overly-cumbersome efforts for self-protection. Often, the lack of a trusted party means that large numbers of nodes must interact according to complex and expensive protocols, the simplest example of which is secret sharing [Sha79, Bla79].

While techniques such as Zero-Knowledge Proofs (ZKP's) [GMR89, GMW86] can ensure that a critical component has followed the steps it is required to perform, they cannot verify that the actions of critical components were *restricted* to the required steps. That is, it is not possible to verify that a trusted component has kept information secret. ZKP's allow one to verify what has been done, but not what hasn't.

Thus, although cryptographic tools can allow a sort of discreet integrity check on TTP's, they cannot ensure that TTP's are discreet. One still needs to trust the third party in a strong manner.

1.4.1 Least Reliance

The principle of *Least Privilege* [SS75] states that users and processes in a system should have the minimal set of access rights needed to accomplish their tasks. This confinement protects the system and other users from abuses, errors, and Trojan Horses.

In a modern internetworking scenario, we have the converse concern, namely to protect users and processes from the actions of third parties. Reversing the roles of operating system and user for a moment, it is now the user who wishes to protect her resources from the actions undertaken by some foreign service.

Least Privilege thus becomes:

> **(Least Reliance)** *When the architecture of a distributed system or service requires that a client trust a service provider, the client should rely on the service provider in the least possible fashion.*

For illustration, consider a tax-return preparation service. If the client must submit her data, then the architecture has demanded that the server be relied upon not to release her data. If, however, the server provides a (non-communicating) applet to the client, the architecture of this solution demands that the server be relied upon only to provide a correct program. Clearly, in the latter case, the client is not relying on the server's discretion

[2] This is itself a strong assumption for information security, particularly when differentiation is limited. But we shall avoid discussing the problems of maliciously monopolistic or market-driven uniformity.

to keep sensitive information private, since the information flow is uni-directional.

Trusted third parties are typically relied upon for *discretion* (privacy) and *integrity* (correctness). In the preceding example, reliance was minimized to simple correctness; leakage of information from the service provider would not compromise the private information of the client.

The two important points to address are:

1. In what settings is it possible to reduce trust placed in third parties?
2. What is the cost of minimizing reliance on third parties?

1.5 From Server Involvement to Server Assistance

The main distinction between *server-assisted* solutions and TTP's is the direction of information flow. TTP's typically manage highly sensitive information for their clients, such as decryption and authentication keys. The *server-assisted* architecture demands that information flow solely from trusted party to clients. Thus, the sensitive transactional information (and transactional powers, such as signing contracts or delivering them unfairly in a one-sided fashion) is never placed in the hands of the service providers.

An equally important demand is that clients be able to compose third-party services in order to dissipate risk. In fact, the third-parties/service-providers need not be aware of one another, let alone be required to interact in some sort of coordinated multiparty computation. Indeed, for most if not all tasks, the service-providers do not need to know whether or how many other service-providers are providing services.

1.5.1 Security Resources

Server-Assisted Cryptography focuses on a new class of cryptographic tools, designed to produce and use security resources that are:

1. *transferable*: service providers can transmit resources to clients, who then use them in a simple way;
2. *composable*: resources from distinct sources can be composed (without involvement of the sources) to reduce weaknesses;
3. *independently-produced*: distinct sources need not have any knowledge of other sources that the clients rely upon.

The main question in considering server-assisted security architectures is whether these properties are achievable, and whether they can be achieved in a feasible and simple manner. This paper discusses general approaches to achieving secure composition of resources in the server-assisted model, as well as particular solutions for fundamental cryptographic and transactional tasks.

1.6 Examples

One-Time Pad As a simple example, consider a lightweight client who does not have the ability to generate high-quality random numbers yet wishes to set up a one-time pad (OTP) with a partner for later use. Instead of sending a weak OTP by trusted courier, the client contracts out to a service provider to have identical CD's full of random bits delivered to him and his partner.

Clearly, the resource is transferable: it can be conveyed by private courier on a CD. More interestingly, it is also composable: if two service providers send CD's, the client and partner need only combine them locally with bitwise exclusive-ors to obtain a new OTP that is as strong as the stronger supplier. Thus, if one supplier uses a linear-congruential number generator to produce "random" bits, while the second uses a Blum-Blum-Shub generator [BBS86],[3] the breakable LCNG will not compromise the privacy of the combined CD's.

In this simple example, the client and his partner do rely on obtaining correct (*i.e.* identical) CD's from the suppliers, but the suppliers are never given the highly-sensitive cleartexts. Because the resources are transferred to the clients, however, the suppliers have no idea of the messages being sent (assuming they have no access to the wires used). Moreover, one supplier gains nothing by having access to the ciphertexts, as long as the other supplier has used high-quality random bits.

The client need only have the simple knowledge of how to use a OTP to encrypt messages. He does not need a stochastic source and/or a strong random number generation package.

[3] Linear congruential number generators are known to be predictable [Plu82], whereas predicting BBS sequences is as hard as factoring [BBS86].

There are a few essential points to note about this example. First, the specialized cryptographic work (generating good random numbers, in this case) have been outsourced to experts, reflecting a division of labor for security tasks themselves. Second, composition increases the strength of the resulting resources. Third, and most importantly, the service providers never see the sensitive data; thus, any reliance on their discreet handling of sensitive cleartexts is obviated.

Remark: Commodities *vs.* Services. The OTP-CD example illustrates a slightly restricted version of server-assisted cryptography, which we have described as *commodity-based* [B97]. The difference is that the resources can be delivered as a single response to a request, and the servers need not be available on-line. That is, the resources can be packaged, delivered, and used much later on.

Joint Computations The OTP example is fairly trivial and suffers from a bootstrap motivation: how would the TTP's communicate securely with the clients in the first place (trusted courier aside)? The problem does bare some teeth when one starts to worry about possible errors in the CD's. And it should be noted that generating good randomness is essential for sending secure messages but not for receiving them. But there turn out to be other areas in which server-assistance provides improved yet nontrivial solutions.

As described in more detail in §4, there are a host of cryptographic tools to allow mutually-distrusting parties to compute some function on their respective inputs without revealing them. A typical example is to compare a password with a password attempt without revealing either, while still obtaining the single-bit answer of whether they match. These tools provide some very elegant primitives for protecting information and interaction, but they remain unused because of their high complexity.

It turns out that simple, server-assisted techniques can be used to provide the functionality of these purely abstract primitives. Many of them can be reduced to a basic cryptographic primitive called Oblivious Transfer (OT), which is essentially a noisy channel with (oddly enough) guaranteed noise that is undetectable by the sender. While existing cryptographic research on OT has churned out "polynomial time" but otherwise highly expensive solutions, we show that a server can assist in achieving OT by providing sender and receiver with a small, easily generated set of quadruples of numbers.

The solution is nowhere near as simple as exclusive-oring sequences of random bits, but despite the somewhat complicated technical justifications, they are drastically simplified in relation to their cryptographic predecessors. Although a solution for a general-purpose function-evaluator still remains moderately complex (from a systems viewpoint), we have implementation evidence that the OT solution is within the reach of fairly lightweight clients.

1.7 Roadmap

In the remainder of this presentation, we specify in more detail what "server-assisted cryptography" requires (§2), discuss some motivations from cryptography (§3), and then give some technically-justified but (ultimately) easily implemented solutions for a couple of central cryptographic problems (§4-5).

2 Definitions

For completeness, we include some more formal definitions and background, some of which can be found in [B97]. A simple, special case of server assistance comes in the form of *commodity-based* tools [B97], in which services can essentially be stored, transferred, and used for later computations without online presence of the service provider.

A **two-tiered (n,m)-protocol** $\Pi = (\mathcal{C},\mathcal{S})$ is a collection of $n+m$ probabilistic interactive Turing Machines (PTM's), divided into two groups, $\mathcal{C}$ (clients) and $\mathcal{S}$ (servers). The clients are poly-time PTM's (PPTM's), and the servers may or may not be restricted to poly-time, depending on circumstances.

Each client has a unique id $i \in \{1,..,n\}$, and each server has a unique id $h \in \{1,..,m\}$. An **execution** of Π on input $\vec{x} = (x_1,\ldots,x_n)$ is the network computation induced by running each client i on input x_i. An execution induces a distribution $\Pi(x) = (y_1,\ldots,y_n)$ on the clients' outputs. Letting $X_\kappa \subseteq \{0,1\}^\kappa$, $Y_\kappa \subseteq \{0,1\}^\kappa$, and $\mathtt{dist}(Y_\kappa^n)$ indicates the set of distributions on Y_κ^n, we may write this as $\Pi : X_\kappa^n \to \mathtt{dist}(Y_\kappa^n)$.[4] If α and β

[4] It is straightforward to generalize this notation to ac-

are distributions with support $S \subseteq (\{0,1\}^\kappa)^n$, define the **distance** between them as $\| \alpha - \beta \| = \frac{1}{2} \sum_{y \in S} |\Pr[\alpha = y] - \Pr[\beta = y]|$. We say that Π **computes** $\hat{\Pi}$ $\epsilon(\kappa)$-reliably if for all κ, for all $x \in X_\kappa^n$, $\| \Pi(x) - \hat{\Pi}(x) \| \leq \epsilon(\kappa)$. Π is a **statistically reliable** implementation of $\hat{\Pi}$ if it computes F, $\kappa^{-\omega(1)}$-reliably. Π is a **computationally reliable** implementation of $\hat{\Pi}$ if for any PPTM Dist, for all κ and $x \in (\{0,1\}^\kappa)^n$, $|\Pr[\mathtt{Dist}(\kappa, \Pi(x)) = 1] - \Pr\left[\mathtt{Dist}(\kappa, \hat{\Pi}) = 1\right]| = \kappa^{-\omega(1)}$.

Let $F = \{F_\kappa\}$ where $F_\kappa : X_\kappa^n \to Y_\kappa^n$. If $\hat{\Pi}$ is a protocol in which a single uncorruptible server collects all inputs x_i and returns the respective components of $F(x_1, \ldots, x_n)$, then Π is said to compute F if Π is a reliable implementation of $\hat{\Pi}$. Further details can be realized through natural generalizations of [B91b, MR91].

The interaction between client and server may be characterized through the following:

Definition 1 *A two-pass protocol between client C_i and server S_h (in which a client C_i generates a request string $q_{h,i}$ and receives a response string $y_{h,i}$ from server S_i) is called a* **stateless oblivious RPC** *(remote procedure call) if $q_{h,i}$ is independent of C_i's input x_i (apart from x_i's length) and of any previous communications with S_h or other service providers (apart from including tags for identifying and authenticating C_i and S_h).*

The "oblivious" nature of the RPC captures the important *information flow property*, which can be described informally as:

> **(Information Flow)** *Private information does not flow from any client to any collection of servers; nor does information flow from one server to another.*

In the simplest form, servers provide resources in the form of commodities that can be purchased and transferred.

Definition 2 *A two-tiered protocol Π is* **commodity-based** *if:*

1. *no communication among servers is necessary;*
2. *servers do not need to know the identities, number, or existence of other servers;*
3. *for each client $C_i \in \mathcal{C}$ and server $S_h \in \mathcal{S}$, C_i interacts with S_h only through stateless oblivious RPC's.*
4. *apart from negotiation of $(\max|x_i|, \mathcal{S}, n, \kappa)$ (namely, maximal client private input size, server ID's, number of servers, and security parameter), interactions among clients that may depend on client inputs $\{x_i\}$ occur strictly after all client-server interactions.*

The commodity-based case of server-assistance is simple, yet robust enough to support several important cryptographic tools.

Ideally, each client-server interaction consists of a single "purchase" of an appropriate commodity. Multiple RPC's are allowed in order to accommodate larger, composite protocols. The protocols presented in this paper require at most a single RPC from a given client to a given server.

It is important to maintain the stateless property to avoid excessive and unscalable demands on servers. We are specifically interested in ensuring that the interaction provides object-like commodities rather than ongoing functional services, and that it does not degenerate into an indirect way for servers to communicate with one another.

2.1 Server Assistance

In more arbitrary settings, however, it may be useful to permit a short-term, stateful interaction rather than a simple RPC. One motivation is to provide greater flexibility in the generation and transfer of the commodities. More importantly, transactional services generally require the online presence (however brief and simple) of the service providers.

We require that this stateful interaction be of bounded duration and avoid violating the information flow rule.

Definition 3 *A two-tiered protocol Π is* **server-assisted** *if:*

1. *no communication among servers is necessary;*
2. *servers do not need to know the identities, number, or existence of other servers;*
3. *for each client $C_i \in \mathcal{C}$ and server $S_h \in \mathcal{S}$, C_i interacts with S_h only through oblivious RPC's.*

commodate a length of each y that is polynomial in κ.

3 Cryptographic Tools

Following the lead of data mining and application mining, we turn to "crypto mining" to see whether there are elegant cryptographic ideas that can be made viable using a server-assisted approach. (This is not a purely self-serving cryptographic exercise, in that many crypto methods support appealing trust-management properties that are otherwise unreachable because of complexity. One example is the process of dissipating risk by relying on an honest majority in large groups.)

Although there is a whole body of cryptographic research for solving tasks beyond the "core" cryptographic operations of key exchange, strong randomness, encryption, signing, and hashing, very little of it has seen the light of day. Indeed, the body of code needed to support just the core operations already strains the limits of "lightweight" implementation. Protocols with higher complexity are generally too complicated to implement efficiently and without subtle error. The server-assisted approach has the potential to simplify them to bring them within the reach of application.

• **Proof Systems.** Proof systems enable one party to demonstrate that the results of its computation (such as an encrypted message) were constructed according to a specified protocol. Zero-knowledge proof systems (ZKPS) in particular allow the prover to protect the actual evidence, such as cleartext, encryption keys, or internal computations. The ability to demonstrate a fact without revealing it is an elegant idea with obvious application, for example, to verifying passwords, or showing that an applicant is in possession of the private key corresponding to the public key he wishes certified.

• **Commitment.** A mechanism for committing information without revealing it immediately is useful for a variety of purposes, such as sealed auctions. It can be used for cryptographic goals such as generating shared random sequences, preventing chosen ciphertext attacks, and implementing zero knowledge proof systems.

• **Secret Sharing and Escrow.** Although key escrow refers to the holding of decryption keys by an external party, one of the arguments in defense of it is that those keys need not be stored at a single site. By splitting them up into shares and distributing them to different sites or organizations, a key escrow mechanism can obtain some small degree of resistance to abuse, by requiring that those shares be obtained (presumably properly) from a majority of sites if the key needs to be determined. Through secret sharing, the information is at the behest of the majority, not any particular individual.

The related (and more appropriate) meaning of "escrow" enjoys less attention in cryptography. There are few mechanisms to hold something of value in escrow as enforcement of a contract; in part, because of impossibility-type results [Cle86]. The primary topic is the fair exchange of keys [Blu83] (not the same thing as "key exchange"), reflecting the idea that each key protects some item of value to be exchanged for the other. Note that in classical "escrow" situations, the escrowing party has complete control of the escrowed item, implying a very strong degree of trust.

• **Joint Computation.** Electronic voting and threshold digital signatures are examples of scenarios in which a value based on private inputs needs to be computed without revealing the inputs. Voters' decisions should remain private, but the tally must be determined. A signed document must be generated if a quorum of company directors ratifies it, but the secret signing key should not be revealed or localized.

In its general form, joint computation is described as applying some function $f(x_1, \ldots, x_n)$ to private inputs x_i, with participant i holding x_i. It is convenient to imagine that there is a (*virtual*) trusted party who receives the inputs on private lines, then reports precisely $f(x_1, \ldots, x_n)$ in return. Two-party ($n = 2$) computations can generalize many of the primitives described above (proof systems, commitment).

• **Oblivious Transfer.** As introduced by Rabin [R81], OT is a two-party protocol by which Alice sends a bit b to Bob, which arrives with probability 1/2. Alice does not learn whether b arrived, however.

This odd but simple mechanism has ubiquitous application within cryptographic methods for joint computation. Protocols for zero-knowledge proofs, commitment, and multiparty computation [K88] can be built on OT as a primitive.

3.1 Technical Results

In the sequel, we focus on achieving oblivious transfer and fair exchange using server-assisted methods. In particular, the data to be transferred or exchanged will never fall in the hands of a third party; rather, the third party will provide simple resources to enact the transaction.

Our motivation is to show that server-assisted architectures have an immediate payoff in cryptography, by simplifying the large number of currently unimplementable protocols. The natural goal is to determine whether these or other changes to the trust models can ultimately enable loosely-coupled systems to capitalize on elegant cryptographic ideas.

4 Virtual Mediation

Many transactions are best described in terms of a TTP who accepts suitable private information from the clients and performs a desired service, such as generating encrypted tickets or exchanging signed contracts. One branch of cryptographic research has investigated how such TTP's can be replaced by interactive protocols which achieve the same results with a high degree of security, yet without placing trust in particular parties [GMW86, GMW87, BGW88, CCD88]. These multiparty protocols provide a *virtual mediator*, an entity that exists only through the action of the protocols.

Regardless of the feasibility or suitability of multiparty protocols, the concept of a non-existent but effectively present mediator is a powerful notion for organizing and designing cryptographic solutions based on *risk-dissipation*.

The server-assisted model supports this kind of risk dissipation but is not constrained to it. That is, it is certainly possible that:

1. There is one server, who provides services that are independent of the clients' inputs;

2. There are several servers, who provide services that are independent of the clients' inputs; these services are combined by the clients to assure that errors in one or more services do not compromise the overall transaction.

In other words, the strategy of replacing one entity by several – in order to dissipate risk, increase integrity, and reduce vulnerability – is compatible with server-assisted cryptography but is not absolutely necessary.

There are several distinctions between the well-studied multiparty protocol approach and the proposed server-assistance model. In particular, in multiparty protocols, the clients are themselves the servers; they are responsible for managing the heavyweight interaction required for simulating a virtual mediator. In the server-assisted approach, the number of clients and the number of servers have no connection at all.

4.1 Reusing Cryptographic Tools

The multiparty approach is suggestive, however, and it can provide the cryptographic basis for many server-assisted techniques. As described below for the example of oblivious transfer, one can first imagine that the servers engage in a multiparty protocol, thereby simulating a non-existent, virtual mediator. But the server-assisted model requires that servers do not interact in such a fashion; so instead, we might arrange for the clients to manipulate the resources they obtained from the servers in such a way that it appears as though the servers had indeed engaged in a multiparty protocol.

That is, through a doubly recursive application of virtual-mediator simulation techniques, it may be possible for clients to simulate interacting servers, who are themselves simulating a virtual mediator who satisfies the tasks that a TTP should perform.

The surprising result is that, for certain tasks, this excessively abstract and convoluted approach turns out to collapse to extremely simple client-client protocols satisfying the demands of the server-assistance model. In other words, the composition and use of resources from multiple servers is ultimately direct and simple, despite the roundabout high-level design.

4.2 Two-Party Transactions on Private Inputs

As an illustration, we recount the commodity-based solution for oblivious transfer of [B97]. Many cryptographic tasks can be accomplished with OT as a primitive, including zero-knowledge proofs, bit commitment (effectively placing a bit in an envelope where it cannot be changed), and secure two-

party computation of any discrete function $f(x, y)$ on private inputs.

Two equivalent variants of OT are helpful. Even, Goldreich and Lempel introduced the notion of one-out-of-two oblivious transfer ($\frac{1}{2}$OT), in which Alice holds two bits, b_0 and b_1, and Bob receives one, uniformly and at random [EGL82]. Alice does not learn whether Bob received b_0 or b_1; Bob's result may be expressed as (c, b_c) for a random $c \in \{0, 1\}$. In chosen one-out-of-two OT ($\binom{2}{1}$OT), Bob chooses c and receives b_c. The natural specification protocol, against which implementations are measured, includes a trusted third party T who accepts Alice's bits (b_0, b_1), Bob's choice c, and returns (c, b_c) to Bob. Crépeau showed equivalences among these variants [C87].

There are a variety of implementations of OT and its variants. The security of each rests on certain assumptions. Some rely on assuming that certain number-theoretic problems are intractable, such as factoring or computing discrete logarithms [R81, BM89]. Others assume that the laws of quantum mechanics hold, thereby providing a physical uncertainty that forms the basis for hiding results from Alice or bits from Bob [BBCS91]. Still others rely on the existence of a majority of honest players in a known, completely-connected network with private communication channels [B87].

We present a two-tiered protocol for oblivious transfer and show that it is commodity-based according to Def. 2. The security of our protocol rests on the existence of a majority of honest servers among the m servers.

4.2.1 Notation

Let $\mathbf{Z}_p$ be the field of integers modulo p for some prime p. If S is a set, we denote taking a uniformly random sample from S by $s \leftarrow \$(S)$.

In describing the protocols, a local computation by party X is written in the form $X : x \leftarrow f(y)$. Sending a message m from X to Y is denoted by $X \rightarrow Y : m$.

4.3 Virtual Mediation by the Servers

As a first approximation, we permit the servers to interact (see the discussion in §4). It is then possible to have the servers apply the secret-sharing-based multiparty protocol solutions of [BGW88, CCD88].

In particular, imagine that Alice and Bob secretly share their inputs (the data bits and the choice bits for OT) among the collection of m servers [Bla79, Sha79]. Let $t < m/2$ be a bound on the number of faulty servers. Using polynomials $f_0(u)$, $g_1(u)$, and $g(u)$, with $f_0(0) = b_0$, $f_1(0) = b_1$, and $g(0) = c$, Alice and Bob provide $f_0(h)$, $f_1(h)$, and $g(h)$ to server h.

At this point, the servers jointly hold b_0, b_1, and c, although any minority cannot determine the values. Together, they calculate a new polynomial of degree $2t$,

$$h(u) = f_0(u)(1 - g(u)) + f_1(u)g(u) + r(u)$$

by individually calculating $h(i) = f_0(i)(1 - g(i)) + f_1(i)g(i) + r(i)$. Here, $r(u)$ is a random polynomial of degree $2t$ with $r(0) = 0$. (This could easily be provided by having Alice and Bob each share a degree-$2t - 1$ polynomial $r_A(u)$ and $r_B(u)$, then setting $r(u) = (r_A(u) + r_B(u))u$.) Thus, $h(0) = b_0(1 - c) + b_1 c = b_c$. Therefore, the servers need only provide their point on h to Bob, who can then derive b_c as desired.

Of course, we will not allow the servers to interact. Instead, Alice and Bob will perform an essentially isomorphic protocol, which turns out to be much simpler.

4.4 Virtual Virtual Mediation

Using quadruples provided by the servers, Alice and Bob will mutually determine the state of *virtual* servers (which "exist" only as the combination of their private information) who perform the $\binom{2}{1}$OT transfer.

For a virtual state variable x, we write $\psi_0(x)$ as the local, private information held by Alice, and $\psi_1(x)$ as the local, private information held by Bob, where together, $\psi_0(x)$ and $\psi_1(x)$ determine x, even though neither Alice nor Bob may know x itself.

By appropriate manipulation of these values, Alice and Bob can pretend as though one virtual server i calculated (for example) $h(i) = f_0(i)(1 - g(i)) + f_1(i)g(i) + r(i)$. Neither Alice nor Bob will know $h(i)$, but together they determine it.

A major obstacle is that the resources provided by the real servers are independent of Alice's and Bob's inputs (as decreed by our model!), thus it is not clear how to tie them to the appropriate values later on.

Server-OT-Program(server: h; input: B = # transfers, κ = security, p = κ-bit prime)

1.1. S_h: for $j = 1..B$

$x_{h,j} \leftarrow \$(0,1)$, $y_{h,j} \leftarrow \$(0,1)$, $z_{h,j} \leftarrow \$(0,1)$

$w_{h,j} \leftarrow x_{h,j} - z_{h,j}x_{h,j} - z_{h,j}y_{h,j}$

$\psi_0(x_{h,j}) \leftarrow \$(\mathbf{Z}_p)$, $\psi_0(y_{h,j}) \leftarrow \$(\mathbf{Z}_p)$,

$\psi_0(z_{h,j}) \leftarrow \$(\mathbf{Z}_p)$, $\psi_0(w_{h,j}) \leftarrow \$(\mathbf{Z}_p)$

$\psi_1(x_{h,j}) \leftarrow x_{h,j} - \psi_0(x_{h,j})$, $\psi_1(y_{h,j}) \leftarrow x_{h,j} - \psi_0(y_{h,j})$

$\psi_1(z_{h,j}) \leftarrow x_{h,j} - \psi_0(z_{h,j})$, $\psi_1(w_{h,j}) \leftarrow x_{h,j} - \psi_0(w_{h,j})$

2.1. $S_h \rightarrow$ A: $\{(\psi_0(x_{h,j}), \psi_0(y_{h,j}), \psi_0(z_{h,j}), \psi_0(w_{h,j}))\}_{j=1..B}$

2.2. $S_h \rightarrow$ B: $\{(\psi_1(x_{h,j}), \psi_1(y_{h,j}), \psi_1(z_{h,j}), \psi_1(w_{h,j}))\}_{j=1..B}$

Figure 1: Security resources provided by server h. (\$() denotes random samples.)

This can be achieved using linear adjustments to the values, a tool first designed and applied in [B91c]. The servers provide resources that can be made to look like secret shares of random numbers (along with their products); thus independence from b_0, b_1, and c is achieved. Later, Alice and Bob communicate with each other – not the servers – to make the appropriate adjustments to these purely random values.

4.4.1 Generation of Commodities

The servers provide extremely simple commodities, namely random quadruples (w, x, y, z) satisfying a simple constraint (within the field of arithmetic used for computation):

$$w = x(1 - z) - zy.$$

Thus, a server just needs a strong random number generator; the actual generation of resources is simple.

4.4.2 Use of Commodities

Figs. 2 and 3 describe how Alice and Bob obtain the commodities and apply them to the desired input bits. The previously-mentioned adjustments appear as corrections $(\Delta x, \Delta y, \Delta z)$ to the (x, y, z) values. These adjustments become publicly known to Alice and Bob, but they are differences between sensitive values and purely random, secret values; thus, they reveal no information.

Despite the algebraic mess of Fig. 3, the computations performed by Alice and Bob are extremely simple: mere linear combinations of resource values.

4.5 Malicious Servers and Malicious Clients

As described, the protocols do not resist malicious attacks, either by service providers who generate faulty commodities or by Alice or Bob. They do survive passive compromise of a minority of servers, as well as "honest-but-stupid" errors, in which predictable random number generators are used by a minority of servers.

Adding robustness against malicious faults turns out to be direct and simple: in addition to providing a sum-shared quadruple (w, x, y, z) to Alice and Bob, a server will also provide Bob with information that commits Alice to her share, and vice versa. A suitable arithmetic commitment scheme that admits exponentially-small chance of error can be found in [RB89].

Second, the generation of the polynomials $f_{0i}(u)$, $f_{1i}(u)$, $f_{2i}(u)$, and $f_{3i}(u)$ must be verified. Even though we face a two-party case, this can be achieved by adapting techniques developed for multiparty secret computation in [RB89] or [B91a], with commodities from a particular source serving as the substrate for evaluating a particular player's computation in a virtual multiparty protocol.

Third, it is necessary to verify that $f_{0j}(0) \in \{0,1\}$, $f_{1j}(0) \in \{0,1\}$, and $f_{2j}(0) \in \{0,1\}$, before the enabling information of step 4.2 is given to Bob. This is done through a computation similar to that described in Fig. 3, by calculating $r_{0j}(u) + f_{0j}(u)(1 - f_{0j}(u))$, $r_{1j}(u) + f_{1j}(u)(1 - f_{1j}(u))$, and $r_{2j}(u) + f_{2j}(u)(1 - f_{2j}(u))$. Here, $r_{0j}(u)$, $r_{1j}(u)$, and $r_{2j}(u)$ are of degree $2t$ with free term 0 and play a role similar to that of $f_{3j}(u)$.

```
Client-OT-Protocol(input: x_A = Alice's bits {(b_0j, b_1j)}, x_B = Bob's choices {b_cj},
      B = |x_A|/2 = # transfers, m = # servers, κ = security, p = κ-bit prime)
0.1.  t ← ⌈m/2⌉ − 1
1.1.  for h = 1..m
                A → S_h:      (B, κ, p)
1.2.  for h = 1..m
                B → S_h:      (B, κ, p)
2.1.  A:        for h = 1..m
                      receive {(ψ_0(x_hj), ψ_0(y_hj), ψ_0(z_hj), ψ_0(w_hj))}_j=1..B
2.2.  B:        for h = 1..m
                      receive {(ψ_1(x_hj), ψ_1(y_hj), ψ_1(z_hj), ψ_1(w_hj))}_j=1..B      (cont.)
```

Figure 2: Client programs to obtain and use OT resources provided by servers $1..n$. (cont.)

5 Fair Exchange

A central task in electronic commerce in decentralized networks is to ensure that payment is given if and only if the purchased goods are provided. That is, a transaction must be *atomic*. Signing a contract is a similar case: if either party obtains the other's signature, then both parties should obtain each other's signature.

For relatively obvious reasons, when there are only two parties, it is difficult (in fact, impossible) to ensure atomicity. For any given protocol, one player can simply withdraw when she has a slight advantage. This scenario has been investigated under the rubrik of "secret key exchange" [Blu83], and Cleve has shown that the advantage obtainable through a simple fail-stop attack inversely proportional to the number of rounds in the transaction protocol [Cle86].

Clearly, a TTP can simply accept the two valuable digital items (payment and goods; or signature and signature), verify them against one another if needed, and then deliver them as desired. In this simple solution, however, the TTP sees all sensitive information, which may be undesirable to Alice and Bob.

We propose a *server-assisted* solution. Let Alice and Bob's task be specified by a function $f(x,y) = (z_A, z_B)$ on inputs x held by Alice and y held by Bob. Alice is to receive z_A if and only if Bob receives z_B.

Instead of applying standard cryptographic techniques to calculate $f(x,y)$ without revealing x to Bob or y to Alice, we have Alice and Bob compute a new function $\hat{f}(x,y)$ that provides verified sum-shares of z_A and z_B.

In particular, consider a verified commitment scheme along the lines of Tompa/Woll and Rabin/Ben-Or [TW87, RB89]. To arrange that Alice holds some private value a, choose a random line with free term a and evaluate it at a secret nonzero point. In other words, choose random b,$c \neq 0$, and d with

$$a + bc = d,$$

and give $\text{rev}(a) = (a,b)$ to Alice. Give $\text{chk}(a) = (c,d)$ to Bob. Clearly, (c,d) gives no information about a.

When Alice wishes to reveal a, she produces $\text{rev}(a)$. To verify this against $\text{chk}(c,d)$, simply check whether $a + bc = d$:

$$\text{Verify}((A,B),(C,D)) = \begin{cases} \text{(yes,a)} & \text{if A + BC = D} \\ \text{(no,0)} & \text{otherwise} \end{cases}$$

The new task $\hat{f}(x,y)$ generates secret random numbers α and β and then outputs:

$$\begin{aligned} \hat{f}(x,y) \quad = \quad & ((z_A + \alpha, \beta, \text{rev}(\beta), \text{chk}(\alpha)), \\ & (\alpha, z_B + \beta, \text{rev}(\alpha), \text{chk}(\beta))). \end{aligned}$$

After computing $\hat{f}(x,y)$ using standard techniques (indeed, a generalized version of the server-assisted OT solution given above is possible), Alice obtains $(z_A + \alpha, \beta, \text{rev}(\beta), \text{chk}(\alpha))$, and Bob obtains $(\alpha, z_B + \beta, \text{rev}(\alpha), \text{chk}(\beta))$.

Alice sends $(\text{rev}(\beta), \text{chk}(\alpha))$, to the service provider. Bob sends $(\text{rev}(\alpha), \text{chk}(\beta))$.

Having received some $((A_1, A_2), (A_3, A_4))$ from Alice and $((B_1, B_2), (B_3, B_4))$ from Bob, the service provider calculates $\text{Verify}((A_1, A_2), (B_3, B_4))$ and $\text{Verify}((B_1, B_2), (A_3, A_4))$. If the results are (yes, A_1) and (yes, B_1), then the service provider

```
Client-OT-Protocol(continued)
3.1.  A:      select random polynomials over Z_p:
                  f_0j(u) of degree t with f_0j(0) = b_0j
                  f_1j(u) of degree t with f_1j(0) = b_1j
                  f_3j(u) of degree 2t with f_1j(0) = 0
3.2.  B:      select random polynomials over Z_p:
                  f_2j(u) of degree t with f_2j(0) = c_j
3.3.  A:      for h = 1..m, j = 1..B
                  Δx_hj ← f_0j(h) − ψ_0(x_hj)
                  Δy_hj ← f_1j(h) − ψ_0(y_hj)
3.4.  B:      for h = 1..m, j = 1..B
                  Δz_hj ← f_2j(h) − ψ_1(z_hj)
3.5.  B→A:    {Δz_hj}_{h=1..m,j=1..B}
4.1.  A:      for h = 1..m, j = 1..B          // v_hj will be f_4j(h)
                  ψ_0(v_hj) ← Δx_hj − Δz_hjΔx_hj + Δz_hjΔy_hj
                        +ψ_0(w_hj) − ψ_0(z_hj)Δx_hj − ψ_0(x_hj)Δz_hj
                        +ψ_0(z_hj)Δy_hj + ψ_0(y_hj)Δz_hj
                        +f_3j(h)
4.2.  A→B:    {ψ_0(v_hj)}_{h=1..m,j=1..B}
5.1.  B:      for h = 1..m, j = 1..B
                  ψ_1(v_hj) ← ψ_1(w_hj) − ψ_1(z_hj)Δx_hj − ψ_1(x_hj)Δz_hj
                        +ψ_1(z_hj)Δy_hj + ψ_1(y_hj)Δz_hj
                  f_4j(h) ← ψ_0(v_hj) + ψ_1(v_hj)
              for j = 1..B
                  interpolate f_4j(u)
                  b_cj ← f_4j(0)
```

Figure 3: Client programs to use OT resources provided by servers $1..m$.

forwards $A_1 = \beta$ (true with high probability) to Bob and $B_1 = \alpha$ to Alice. Finally, Alice calculates $z_A = z_A + \alpha - B_1$, and Bob calculates $z_B = z_B + \beta - A_1$.

It is not hard to see that the values received by the third party are independent of z_A and z_B (barring willful misbehavior by Alice or Bob, of course). Thus, this solution meets the information flow property.

It should be noted that a malicious service provider can, besides denying one player a valid output, cause Alice or Bob to accept an incorrect output. The protocol can be enhanced to ensure that such attempts cause the deceived client to realize that either the service provider or the other party has misbehaved – but it is impossible to detect which. By using more than one service provider, however, Alice and Bob can be assured that their received values are correct and were fairly exchanged. Further details can be found in [B96].

6 Concluding Remarks

Security in large-scale, loosely-coupled systems must take into account several important properties, including scalability, efficiency, simplicity, specialization, replication, compartmentalization and local autonomy, differentiation, increasing functionality, and translation. The classical security approach of relying on a trusted computing base (whether it be an operating system kernel or a trusted third party) does not suit this environment. Worse, there are no reasonable cryptographic tools to fill the gap.

Unconstrained environments tend to evolve specialized individuals who provide efficient solutions for particular tasks. The only pressing reason why security might be an exemption to such evolved architectures is that current mechanisms tend to require all-or-nothing trust. This work has focused on moving beyond such simple trust relationships, promoting the principle of

"least reliance" (to evolve beyond trusted computing bases) and suggesting that servers provide assistance without requiring full information.

The technical challenges lie in creating sufficiently simple cryptographic tools to support such an architecture, and in analyzing the diverse and possibly unexpected trust relationships that will arise as a result. For many cryptographic tasks that are otherwise out of reach for reasons of complexity, the server-assisted approach provides drastically simplified mechanisms that demonstrate the feasibility of those tasks under new trust relationships.

Acknowledgements

The author would like to thank Cathy Meadows for many helpful comments during the extensive revision of this paper (all views and errors remain the responsibility of the author, of course!), and Steven Greenwald and Mary Ellen Zurko for their patient attention during the review process.

References

[B87] D. Beaver. "Oblivious Secret Computation." Harvard University TR-12-87, December 1987.

[B91a] D. Beaver. "Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority." *J. Cryptology* **4**:2, 1991, 75–122.

[B91b] D. Beaver. "Foundations of Secure Interactive Computing." *Advances in Cryptology – Crypto '91 Proceedings,* Springer–Verlag LNCS 576, 1992, 377–391.

[B91c] D. Beaver. "Efficient Multiparty Protocols Using Circuit Randomization." *Advances in Cryptology – Crypto '91 Proceedings,* Springer–Verlag LNCS 576, 1992, 420–432.

[B95] D. Beaver. "Precomputing Oblivious Transfer." *Advances in Cryptology – Crypto '95 Proceedings,* Springer–Verlag LNCS 963, 1995, 97–109.

[B96] D. Beaver. "Fair Exchange Agents." Manuscript, 1996.

[B97] D. Beaver. "Commodity-Based Cryptography." *Proceedings of the* 29^{th} *STOC,* ACM, 1997, 446–455.

[BM89] M. Bellare, S. Micali. "Non-Interactive Oblivious Transfer and Applications." *Advances in Cryptology – Crypto '89 Proceedings,* Springer–Verlag LNCS 435, 1990, 547–557.

[BBCS91] C. Bennett, G. Brassard, C. Crépeau, M. Skubiszewska. "Practical Quantum Oblivious Transfer." *Advances in Cryptology – Crypto '91 Proceedings,* Springer–Verlag LNCS 576, 1992, 351–366.

[BGW88] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 1988, 1–10.

[Bla79] G. R. Blakley. "Safeguarding Cryptographic Keys." *Proceedings of AFIPS 1979 National Computer Conference,* NY **48**, 1979, 313–317.

[Blu83] M. Blum. "How to Exchange (Secret) Keys." *ACM Trans. Comput. Sys.* **1**:2, May 1983, 175–193.

[BBS86] L. Blum, M. Blum, M. Shub. "A Simple Unpredictable Pseudo-Random Number Generator." *SIAM J. on Computing* **15**:2, May 1986, 364–383.

[BC89] G. Brassard, C. Crépeau. "Sorting Out Zero-Knowledge." *Advances in Cryptology – Eurocrypt '89 Proceedings,* Springer–Verlag LNCS 434, 1990, 150–154.

[BCC88] G. Brassard, D. Chaum, C. Crépeau. "Minimum Disclosure Proofs of Knowledge." *J. Comput. Systems Sci.* **37**, 1988, 156–189.

[CCD88] D. Chaum, C. Crépeau, I. Damgaard. "Multiparty Unconditionally Secure Protocols." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 1988, 11–19.

[Cle86] R. Cleve. "Limits on the Security of Coin Flips when Half the Processors are Faulty." *Proceedings of the* 18^{th} *STOC,* ACM, 1986, 364–370.

[C87] C. Crépeau. "Equivalence Between Two Flavours of Oblivious Transfers." *Advances in Cryptology – Crypto '87 Proceedings,* Springer–Verlag LNCS 293, 1988, 350–354.

[Cr96] C. Crépeau. "Efficient Cryptographic Protocols Based on Noisy Channels." Manuscript, April 9, 1996.

[CK88] C. Crépeau, J. Kilian. "Achieving Oblivious Transfer Using Weakened Security Assumptions." *Proceedings of the* 29^{th} *FOCS,* IEEE, 1988, 42–52.

[DDFY94] A. DeSantis, Y. Desmedt, Y. Frankel, M. Yung. "How to Share a Function Securely." *Proceedings of the* 26^{th} *STOC,* ACM, 1994, 1994, 522–533.

[PSL80] M. Pease, R. Shostak, L. Lamport. "Reaching Agreement in the Presence of Faults." *JACM* **27**:2, 1980, 228–234.

[EGL82] S. Even, O. Goldreich, A. Lempel. "A Randomized Protocol for Signing Contracts." *Comm. of the ACM* **28**:6, 1985, 637–647.

[FFS88] U. Feige, A. Fiat, A. Shamir. "Zero Knowledge Proofs of Identity." *J. Cryptology* **1**:2, 1988, 77–94.

[GMR89] S. Goldwasser, S. Micali, C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems." *SIAM J. on Computing* **18**:1, 1989, 186–208.

[GMW86] O. Goldreich, S. Micali, A. Wigderson. "Proofs that Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design." *Proceedings of the* 27^{th} *FOCS,* IEEE, 1986, 1986, 174–187.

[GMW87] O. Goldreich, S. Micali, A. Wigderson. "How to Play Any Mental Game, or A Completeness Theorem for Protocols with Honest Majority." *Proceedings of the* 19^{th} *STOC,* ACM, 1987, 218–229.

[SNS88] J. Steiner, B.C. Neumann, J. Schiller. "Kerberos: An Authentication Service for Open Network Systems." *Usenix 1988 Conference Proceedings,* 1988, 183–190.

[K88] J. Kilian. "Founding Cryptography on Oblivious Transfer." *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 1988, 20–29.

[Plu82] J. Plumstead. "Inferring a sequence generated by a linear congruence." *Proceedings of the* 23^{rd} *FOCS,* IEEE, 1982, 153–159.

[MR91] S. Micali, P. Rogaway. "Secure Computation." *Advances in Cryptology – Crypto '91 Proceedings,* Springer–Verlag LNCS 576, 1992, 392–404.

[R81] M.O. Rabin. "How to Exchange Secrets by Oblivious Transfer." TR-81, Harvard, 1981.

[RB89] T. Rabin, M. Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority." *Proceedings of the* 21^{st} *STOC,* ACM, 1989, 73–85.

[SS75] J. Saltzer, M. Schroeder. "The Protection of Information in Computer Systems." *Proc. IEEE* **63**:9, September 1975, 1278–130.

[Sha79] A. Shamir. "How to Share a Secret." *Communications of the ACM,* **22**, 1979, 612–613.

[TW87] M. Tompa, H. Woll. "Random Self-Reducibility and Zero-Knowledge Proofs of Possession of Information." *Proceedings of the* 28^{th} *FOCS,* IEEE, 1987, 472–482.

[Y82] A. Yao. "Protocols for Secure Computations." *Proceedings of the* 23^{rd} *FOCS,* IEEE, 1982, 160–164.