Secure protocols for accountable warrant execution

Joshua A. Kroll
kroll@cs.princeton.edu

Edward W. Felten
felten@cs.princeton.edu

Dan Boneh
dabo@cs.stanford.edu

Abstract

We describe cryptographic protocols for secure execution of warrants or legal orders authorizing access to data held by private parties. Using cryptography enables a better combination of security, privacy, and accountability properties than would otherwise be possible. We describe a series of protocols, based on different assumptions about trust and technical sophistication of the parties, and making use of well-studied cryptographic tools. We report benchmark results from our prototype implementation of the tools involved in one such protocol, and show that the protocol's entire computational cost is easily feasible even for very large data sets, such as "cloud" software service or telecommunications databases comprising billions of records.

1 Introduction

It is common for law enforcement or intelligence agencies to require legal clearance, such as a court-issued warrant or order, to get access to data about individuals. We can view this as a multi-party protocol involving the agencies, the court, the data source, and possibly other parties. In this paper we consider how to use cryptography to carry out this type of protocol while providing strong security and privacy properties.

To introduce the parties involved, consider the following scenario: an *investigator* wants to get access to data from some *data source* pertaining to some *identifier*. The investigator communicates with a *court*, and provides the identifier describing the particular data to be retrieved, along with evidence that a legal standard required for access has been met. If the court agrees, it issues an *order* authorizing the investigator to receive records about the specified identifier. On seeing the valid order the investigator is given access to records from the data source relating to the specified identifier.

Although we describe the abstract protocol using terms such as "court" and "order", we emphasize that the protocol applies to a wider variety of cases. We describe here three example uses.

Example 1: Law-enforcement or intelligence agency access to phone records. In this example, an intelligence analyst (the investigator) is given access to a phone carrier's (the data source) records of a targeted person (the identifier) when a special intelligence court (the court) finds (the order) that the analyst has met a legal standard, such as Reasonable Articulable Suspicion, allowing access.

Example 2: Court order for access to business records. In this example, a police department (the investigator) goes to a court (the court) to get an order (the order) for access to business records held by a corporation, or by a third-party software service provider (the data source), and relating to individuals within the corporation (the identifiers) who are under investigation.

Example 3: Cloud provider access to customer data. In this example, an employee of a cloud provider (the investigator) gets formal clearance (the order) from the cloud provider's General Counsel (the court) to access protected data of a specific customer (the identifier) on a cloud server (the data source), in order to investigate possible abuse.

1.1 Goals

Orders and warrants are conventionally executed using straightforward protocols that lack many important security, privacy, and accountability properties. Our goal is to use cryptography to create protocols that offer stronger guarantees.

For example, suppose we want to be able to execute an order without the data source learning which individual was the subject of the order. Obvious ways to achieve this all create security problems. We could try sending the investigator all of the data records (or providing real-time access to the records stored elsewhere), and telling the investigator to access only the one record covered by the order; but then the investigator could access many records undetectably, without an order. Alternatively, we could try having the court give the investigator a secret key allowing access to the covered record; but this would require secure storage of secret keys by the court, and if any of these keys were lost or compromised, then we would have no way to verify what data records were leaked.

In the main body of this paper, we describe a series of protocols that use cryptographic tools to surmount these obstacles, providing more robust security guarantees than are typically found in systems currently used in practice.

1.2 Parties in the system

In the protocols we describe, there are six types of parties (see Figure 2):

- S_i: *data sources*, such as telecommunications carriers or cloud service providers. Data source S_i collects data records $x_{i,j,t}$, where i identifies the data source (e.g., the name of a cloud service or telecommunications provider); j identifies the data subject (e.g., an individual's name, phone number, or email address); and t indicates the time interval, such as the date, at which the data record was collected. We refer to (i, j, t) as the *tag* identifying a data record, and use $x_{i,j,t}$ to refer to the data record itself. (We note that in order to prevent attacks based on traffic analysis and data volume, it may be necessary for each data source to record one data record, possibly an empty one, for every possible user j and for every possible time interval t.)

- I: the *investigator* that seeks legally authorized access to data records.

- C: the *court* or other entity that may approve the investigator's access requests, according to a specified legal standard.

- J: the *judge*, working as part of the court, who approves or denies requests. (In some cases, for clarity, we consider the judge J and the court C to be the same party in the system.)

- D_k: *decryption authorities* who allow the investigator to decrypt the encrypted records after court approval. In order to decrypt each encrypted record, the protocol requires the participation of *every* decryption authority (or a pre-specified fraction, depending on the protocol parameters). In real-world deployments, some of these decryption authorities may be run by parties in the protocol, such as the court, while others may be disinterested third parties. The requirement of these decryption authorities is intended to mitigate the possible loss or compromise of individual secret keys (since multiple secret keys, one for each decryption authority, will now be required for decryption); as well as to provide accountability via external involvement throughout the protocol.

- L: the *auditor*, who maintains a log of the protocol execution. Before proceeding with ordinary protocol steps (such as the court's approval of a warrant request, or the decryption of an approved record), each party will confirm the current step with the auditor L, and wait for confirmation that it has been logged.

  In addition, depending on the particular protocol, the parties may send additional information to L, to enable further third-party and/or public audit. Some of this additional information may not be

sensitive, and can be logged directly to L; while some may be secret, such as the identities of individuals who are the subjects of warrants, and thus must be encrypted before being sent to L.

Whenever auditing information needs to be encrypted, we require the parties to encrypt it using a public-key (asymmetric) encryption system, to a public key pk_L which we call the *escrow key*. The corresponding secret key, sk_L, needed to decrypt these audit log entries, must be a carefully-guarded secret in real-world deployments. It is not needed during the ordinary course of operation, but if security breaches or misconduct are suspected, then it may be retrieved, and used to decrypt and examine the sensitive components of the audit log.

In this case, we distinguish between *after-the-fact auditing* checks, which require the auditor to obtain the secret escrow key, sk_L, and use it to decrypt the sensitive audit log entries; and *on-line auditing* checks, which can be performed during the protocol's execution, using only public information available to the auditor, and serve to ensure that the information encrypted under the escrow key is accurate, and that it will provide as thorough an auditing trail as possible in the event of subsequent after-the-fact audit. In designing secure protocols that provide accountability, an important objective is to ensure that as many security guarantees as possible can be confirmed via auditing checks, with *on-line* checks preferred over checks that can only be done after-the-fact.

Throughout the paper we assume that all parties in the system communicate over confidential and mutually authenticated channels, for example, as provided by the TLS protocol [15].

1.3 Desired security properties

We now give an intuitive overview of several desired security properties, which we will refer to in the body of the paper as we present each proposed protocol. To begin with, during ordinary operation of the protocol, we would like to make sure of the following:

- **Secrecy of the data records**: Nobody learns the content of any data record, except the data source who holds that record, and, if there is a court-approved order for precisely that data record, then also the investigator.

- **Accountability of the requests**: If any party (other than the original data source S_i) learns any data record $x_{i,j,t}$, then the audit log must contain (encrypted under the escrow key) both: (1) a request for that record, cryptographically signed by the investigator's key; and (2) a corresponding order for that record, cryptographically signed by the court's key.

  We remark that these encrypted entries in the audit log can also serve as "sealed" cryptographic commitments by the court to the warrant's contents: if desired, when the court encrypts its signed warrants under the escrow key, it can retain information that will later permit it to "unseal" the encrypted audit log records, revealing the underlying warrant that it originally encrypted. Unsealing an encrypted audit log order in this way will always yield the exact order which the court initially "sealed". Likewise, the analogous audit log entries from the investigator can serve as "sealed" commitments to the investigator's requests.

  This guarantee means that during the ordinary execution of the protocol, even without obtaining the secret escrow key to decrypt any entries, anyone will be able to check the log and see exactly how many orders have been issued (and thus how many corresponding data records have been decrypted). The guarantee also implies that even if the court's or investigator's keys were lost or compromised, the keys still could not be used to leak unauthorized records without leaving a trail of this activity in the audit log.

- **Secrecy of the requests**: Nobody except the investigator and the court should learn which identifiers $(i, j, t$—i.e., the data source, the subject, and the time stamp) were requested by the investigator, unless an audit is deemed necessary and the auditor provides the secret escrow key sk_L and/or asks parties to "unseal" some of the commitments.

Most of these security properties will depend on cryptographic assumptions: for example, that parties cannot decrypt encrypted messages without knowledge of the secret decryption key. These are standard assumptions, which we discuss in detail in the body of the paper.

In addition, every security property will depend on at least some of the parties being *honest* participants: i.e., adhering to the protocol and protecting their secret keys. For example, if the investigator, the court, and all the decryption authorities are dishonest and collude with each other, they can access records without a warrant and without leaving a trail; but this is very unlikely if the decryption authorities are chosen wisely. We also cannot prevent a party who knows information from publishing that information or giving it to a third party. For example, if the investigator receives a record pursuant to an order, no protocol can prevent the investigator from publishing that record. There is no choice but to rely on legal prohibitions to deter such behavior.

In real deployments of secure systems, however, we must contend with the fact that multiple parties may fail to adhere to the protocol, or their computer systems may be compromised or secret keys leaked. Ideally, a secure system should include redundant measures, so that even under such circumstances, some security guarantees are preserved. We briefly enumerate a few of these desired robustness guarantees:

- **Robustness to compromise of data sources**: The risk of loss or compromise of secret keys is particularly apparent at the data sources S_i, since they will be processing and encrypting large streams of data records over long periods of time. Thus, we would like it to be the case that the data source S_i maintains no long-term secret keys, so that a compromise of a data source at one particular time does not also risk leaking data records from past or future data streams. In all our designs, the data sources encrypt data records using *public-key* encryption schemes (rather than *symmetric-key*), which do not require them to maintain any long-term secret keys.

- **Robustness to partial compromise of decryption authorities**: We assume that the secret keys held by some decryption authorities D_i may be compromised or lost. As long as only a small number of authorities are affected there should be no impact to the security and availability of the system. The exact parameters, determining the trade off between how many failures are tolerated and how many of the decryption authorities we must trust, is a matter of policy.

- **Independence of the auditor**: Clearly, the security of the auditor L is paramount for accountability. However, even if L is not participating correctly, the system should still provide as many security guarantees as possible. In particular, even without the correct checks by the auditor, we would like the protocol to guarantee secrecy of the data records and requests, as described above (assuming that the other parties are participating correctly, and that other secrets, including the secret escrow key, have not also been leaked).

- **Rate limiting**: If the secret keys of the court C are compromised, then the investigator may learn the contents of arbitrary records, in lieu of those that are actually approved by a judge. However, we require that the *total number* of data records leaked still matches the number of court approval entries in the audit log.

  In addition, even if the auditor L and the court C are *both* compromised, we would like protocol to limit the amount of information leaked. That is, even in such an extreme scenario, no investigator should be able to learn the contents of the entire database; rather, the rate at which items are retrieved should be capped by the decryption authorities D_i, assuming a subset of them still function correctly.

Finally, we wish to set up additional operational safeguards that would protect the data even in the event that the cryptography used in the protocol fails entirely. The challenge in achieving this stems from the fact that our protocols, described below, operate by encrypting each record with a separate encryption key, and giving all of the resulting ciphertexts to the investigator. When an order is issued, the investigator will be able to recover the secret key needed to decrypt the individual records covered by the order. If the investigator possessed sophisticated technical capabilities that enabled it to bypass the cryptography, it would have the ability to decrypt and access all the records. We would like protocols that are fully secure

under standard assumptions about the investigator's power, and *limit the damage* in extreme cases where those assumptions fail. In other words, even in the event of a failure of cryptography, the protocol should provide security no worse than standard approaches used in practice. Therefore, as an additional safeguard we propose the following security property:

- **Information-theoretic ("Shannon") security**: All encrypted records are held by an intermediary (say, a "data custodian", U), who will only reveal one encrypted record to the investigator for each order issued and signed by the court. With this enhancement, the security is as good as standard systems used in practice today, even if the investigator can entirely break the cryptosystems used to protect records. In Section 6, we discuss the details of how to adapt our protocols in this way, as well as outlining approaches to implementing the "data custodian" securely using untrusted parties.

1.4 Our results

We present five protocols with different security guarantees and varying levels of complexity. We start with two simple protocols that achieve many of our goals, but require the court to maintain a secret key that controls access to plaintext records. Even if we rely on tamper-resistant hardware at the court, this centralized trust in a single component violates our design principles and precludes many desired security guarantees.

We next describe a third protocol that achieves the same functionality as the first two, but replaces the trusted hardware module with a set of decryption authorities that can be run by different organizations—some of whom may be parties in the protocol, such as the court, and some of whom may be independent, disinterested third parties. All decryption authorities (or a pre-specified fraction, depending on policy) must participate in every decryption request approved by the court C, so that there is no single point of failure. The downside is that the decryption authorities D_i must see all of the investigator's requests in the clear. This may be undesirable, especially if some of the decryption authorities D_i are independent third parties who are not authorized to see the investigator's activity.

Our fourth and fifth protocols provide significantly better security properties, though they require more powerful cryptographic tools. Like the third protocol, they make use of multiple decryption authorities so that there is no single point of failure; but in these final two protocols, the decryption authorities are prevented from seeing which data items are being requested by the investigator. The authorities can only verify that (i) the requests are approved by the court C and (ii) the requests are properly logged by the auditor L. Our construction in these last two protocols builds on a protocol for *adaptive oblivious transfer*, due to Green and Hohenberger [19], which is well suited for these settings. We extend this protocol to a new notion, which we call *auditable oblivious transfer*, and make use of this new notion in our fifth and final protocol, which provides the best security of the five, including strong on-line auditability guarantees.

2 A simple design requiring trust in the court's computer systems

We begin with a simple design that gives the court C complete control over access requests. The system assumes that the court's computer systems can maintain a sensitive secret key sk_* and fully control access to and use of this key. This assumption is the primary weakness of this design, as discussed below.

Conventions. For clarity, we will describe all of our protocols without the presence of a "data custodian" U who holds the ciphertexts, and serves them to the investigator when a successful request is made. However, we emphasize that such a party should participate in any real-world instantiation of the protocols; without such a party, our protocols cannot achieve the critical "Shannon" security property of Section 1.3. We discuss how to add a data custodian to these protocols in Section 6.

Preliminaries. In all of the protocols we present below, we will make use of standard cryptographic primitives:

- *Secure channels.* Throughout the paper we assume that all parties in the system communicate over confidential and mutually authenticated channels, for example, as provided by the TLS protocol [15].

- *Digital signatures.* Any secure digital signature scheme is sufficient for our purposes here. Recall that a digital signature scheme has three algorithms: key generation that outputs a secret signing key and a public verification key, a signing algorithm that uses the signing key to sign messages, and a verification algorithm that uses the verification key and a message to confirm that a given signature is valid.

- *Labeled public-key encryption.* We will need a standard public-key encryption scheme. The scheme must be chosen-ciphertext secure and support labeled encryption [28]. Recall that such a scheme has three algorithms: key generation that outputs a public encryption key pk and a secret decryption key sk, a labeled encryption algorithm $\mathrm{ct} := \mathrm{Enc}(\mathrm{pk}, \mathrm{label}, m)$, and a decryption algorithm $\mathrm{Dec}(\mathrm{sk}, \mathrm{label}, \mathrm{ct})$.

  The label used in encryption and decryption is a public value bound to the ciphertext. It provides context for the decryptor. If a ciphertext ct is created with a label i then decryption will fail if one tries to decrypt ct with a different label $i' \neq i$. We note that the assumption of labels is only a convenience, as any chosen-ciphertext secure public-key scheme can be extended to support labels [28].

Protocol 1 (simple system). The system works as follows:

- *Setup.* The system is initialized as follows:

  1. The court C generates a public/private key pair $\mathrm{pk}_*, \mathrm{sk}_*$ for a public-key encryption system as well as a signing/verification pair for a signature scheme. The public-key pk_* is sent to all data sources S_i while the secret key sk_* is kept secret by the court C. The signature verification key is sent to the auditor L.
  2. The auditor L generates a public/private key pair $\mathrm{pk}_\mathrm{L}/\mathrm{sk}_\mathrm{L}$ for a public-key encryption system and a similar such pair for a signature scheme. The log server L sends pk_L and its public verification key to the court C.

- *Normal operation.* All data sources S_i send to the investigator I all their data records $x_{i,j,t}$, each encrypted under the public key pk_*. Here (i, j, t) is the data identifier as explain in Section 1.2. We let

  $$\mathrm{ct}_{i,j,t} := E(\mathrm{pk}_*,\ (i,j,t),\ x_{i,j,t})$$

  denote the ciphertexts sent to I. Recall that this notation means that the data $x_{i,j,t}$ is encrypted using the label (i, j, t) under the public-key pk_*.

- *Investigator queries.* When the investigator needs to decrypt a particular ciphertext $\mathrm{ct}_{i,j,t}$ it sends a signed request to the court C requesting it to decrypt $\mathrm{ct}_{i,j,t}$. The court approves or denies the request. The court sends the request and all associated data to the log server L, encrypted using the log server's public key pk_L and the ciphertext is signed by the court's signing key. We refer to the contents of this log message as the audit record.

  Next, if the request is approved the court uses its secret key sk_* to decrypt $\mathrm{ct}_{i,j,t}$ using the label (i, j, t). The label ensures that the court is decrypting the correct ciphertext, namely the ciphertext associated with identifier (i, j, t). It ensures that the investigator cannot submit a ciphertext associated with some other (i', j', t') and pretend that this ciphertext is for (i, j, t). Finally, the court sends the resulting plaintext back to the investigator.

Remark 1. A small technical issue is that in the scheme above the court learns the plaintext $x_{i,j,t}$ where as in Section 1.3 we required that only the investigator I learn the plaintext. This is easily corrected by having the data sources S_i double encrypt the plaintext, once under pk_* and once under an investigator's public-key. Once the court decrypts the ciphertext it does not learn $x_{i,j,t}$, but the investigator can now recover the plaintext $x_{i,j,t}$, as required.

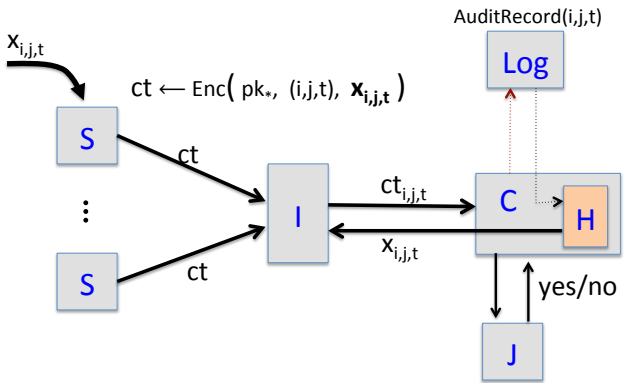

Figure 1: Hardware-based system. Each party S_i is a data source (e.g. phone carriers, email service providers) that encrypts data items $x_{i,j,t}$ using the public-key pk_*. Party I is the investigator, who will make decryption requests to access data items. Party C is a court receiving a decryption request for ciphertext $\mathrm{ct}_{i,j,t}$ from I. Party J is the judge that approves or denies the request. The hardware module H contains the secret key sk_* and responds to a decryption request only if the request is accompanied by a signature from the court and a signature from the log server L to indicate that an audit record for this request has been logged.

Weaknesses. This simple system satisfies many of the requirements listed in the previous section. However, a major weakness of this scheme is that it fully trusts the court's computer systems. If the court's computers are compromised and sk_* is exposed then the security of the entire system is undone and all encrypted records can be decrypted by the attacker. Similarly, an attacker who obtains unrestricted oracle access to the key sk_* can decrypt as many records of its choice without leaving any trace.

Another difficulty with this system is that there is no logging enforcement. Normally, all queries are logged, but if the court's computer systems are compromised it may be possible to decrypt some records without logging the request at the log server L.

A hardware-based mitigation. One way to mitigate the risks described above is to embed the court's secret key sk_* in a tamper resistant hardware such as IBM's CryptoCard secure co-processor [20]. The hardware resides at the court and interacts with its environment through a specific interface described below. If the hardware H is ever tampered with or given invalid inputs it destroys itself and erases its internal memory including the secret key sk_*. The system looks as in Figure 1.

The hardware-based system we describe ensures that the key sk_* is never used unless (i) the court signs the request, and (ii) the log server L confirms that the request is properly logged. This way unauthorized decryptions will be discovered by an audit, assuming we trust that sk_* cannot be extracted from the hardware. Moreover, in the event that the court's signing key is compromised by an attacker, that attacker can issue decryption requests to the hardware H for any ciphertext $\mathrm{ct}_{i,j,t}$ of its choice (and that cannot be avoided), but those requests will all be properly logged by the log server and discovered during an audit.

Protocol 2 (hardware-based system). Using the tamper resistant hardware, denoted by H, a query from the investigator for record (i, j, t) is processed as follows:

1. We assume the hardware H maintains a counter ctr that is set to 0 when the system is first setup. The hardware increments the counter every time it interacts with its environment. This counter is never decremented and is used to prevent replay attacks on H. We also assume that at setup time H is given the log server's public encryption key pk_{L}, the log server's signature verification key, and the court's signature verification key.

2. Once the court C approves a query for record (i, j, t) from the investigator it first reads the current state of the counter ctr from H.

3. The court C then sends the ciphertext $\mathrm{ct}_{\mathrm{C}} := \mathrm{Enc}\big(\mathrm{pk}_{\mathrm{L}},\ \mathsf{ctr},\ (i,j,t)\big)$ to the log server encrypted under the log server's public key pk_{L} (here the label is set to ctr).

   The log server records the request and responds with a digital signature σ_{L} on ct_{C}. The data sent to the log server can include all other data associated with the investigator's request encrypted under pk_{L}, but as a separate ciphertext.

4. Next, the court C sends to the hardware H the data

$$\Big(\ (i,j,t),\ \mathrm{ct}_{i,j,t},\ \mathrm{ct}_{\mathrm{C}},\ \mathsf{rand}_{\mathrm{C}},\ \sigma_{L},\ \sigma_{C}\ \Big)$$

   where $\mathsf{rand}_{\mathrm{C}}$ is the randomness used when creating ct_{C}. σ_{C} is the court's signature on the entire request.

5. The hardware H checks that σ_{L} and σ_{C} are valid signatures, that ctr is the current value of its internal counter, and that ct_{C} is indeed the result of encrypting (i,j,t) under pk_{L} using $\mathsf{rand}_{\mathrm{C}}$ and label ctr. If all these checks succeed the hardware H uses the secret key sk_{*} to decrypt $\mathrm{ct}_{i,j,t}$ using the label (i,j,t). The label ensures that H is decrypting the correct ciphertext, namely the ciphertext associated with (i,j,t). Next, it re-encrypts the plaintext under the investigator's public encryption key and sends the resulting ciphertext to the court's computer.

6. The court sends the returned data from H to the investigator. The investigator decrypts and obtains $x_{i,j,t}$ in the clear.

If we believe that the hardware H can never be coerced into deviating from the protocol above then all decrypted records will be properly logged on the log server L. Even if the hardware H is stolen and operated in a malicious environment it will not be possible to decrypt records without having that action logged on the log server. Of course, if the tamper resistance assumption is violated and a sophisticated attacker can extract sk_{*} from the hardware H then all security is lost.

In the next section we describe a solution that does not rely on trusted hardware, but instead relies on distributing the key sk_{*} across multiple organizations.

3 Eliminating a single point of trust

The system described in the previous section puts considerable trust in the court's computer systems. The risk is that a single organization holds the decryption key sk_{*} putting the entire system in danger if that organization fails to protect the key.

Our solution is to introduce a set of decryption authorities $D_1, \ldots, D_n$ that hold shares of a secret key that functions as sk_{*} (see Figure 2). The authorities can be distributed across multiple organizations so that a compromise of several authorities cannot bring down the security of all records managed in the system. Multiple authorities must be involved in a successful decryption of an encrypted record.

Most likely one decryption authority will reside at the court and that authority can be given special weight: it must participate in every decryption request. We discuss policies of this nature, where some authorities are required to participate, and how to implement them efficiently in Section 6. Another decryption authority could reside at the investigator. Other authorities can include the data sources S_i and possibly other, disinterested third-party organizations.

Decryption authorities. We differentiate between two types of authorities:

- **Cleared authorities** are decryption authorities who are cleared to see the contents of the investigator's query in the clear. The court C, for example, would function as a cleared authority.

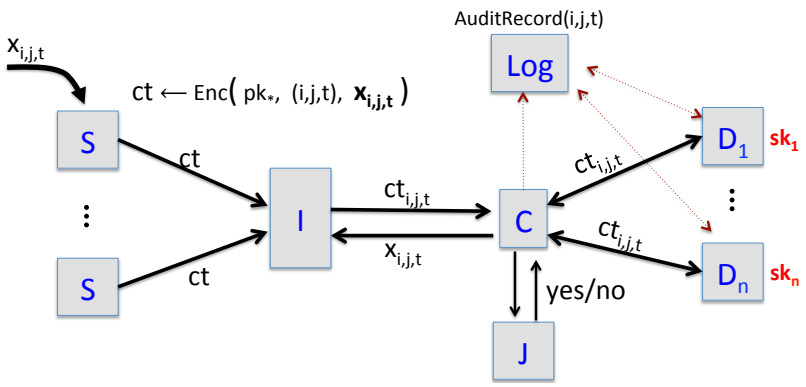

Figure 2: Using cleared decryption authorities. The parties involved are as in Figure 1 except that now the secret key sk_* is shared among n cleared decryption authorities. These authorities are needed for every decryption request and they help ensure that every request is properly logged at L.

- **Uncleared authorities** are decryption authorities who are not allowed to see the contents of the investigator's query in the clear. For example, the system may use some of the data sources as uncleared authorities. These authorities ensure that the court signs every request, but do so without seeing the requests in the clear.

In this section we consider the case where all the decryption authorities are cleared authorities. In the next section we show how to use a mix of cleared and uncleared authorities.

3.1 The system when all decryption authorities are cleared

This case is relatively easy. We use a similar mechanism to the one in the previous section, except that now the secret key sk_* is shared across the decryption authorities $D_1, \ldots, D_n$, as shown in Figure 2.

Cryptographic primitives. To make this work we need a public-key encryption system that supports *threshold decryption* [14]. In such a system the decryption key sk_* is shared across multiple decryption authorities $D_1, \ldots, D_n$. To decrypt a ciphertext $\text{ct}_{i,j,t}$ the court C sends the ciphertext to all active authorities. Each authority applies its share of the secret key and responds with a "partial" decryption of $\text{ct}_{i,j,t}$. The court C combines the partial decryptions into a complete decryption of $\text{ct}_{i,j,t}$. The number of decryption authorities and the subsets authorized to decrypt is a matter of policy.

The public-key encryption system must be chosen-ciphertext secure and support labels as in the previous section. Several such schemes support threshold decryption [29, 3, 9]. The system of Shoup and Gennaro [29] is especially well suited for our purposes.

Protocol 3 (threshold decryption system). The system is similar to Protocol 1. We describe the main differences.

- *Setup.* the system is initialized as in Protocol 1, but now the public/private key pair pk_*, sk_* is generated for a public-key encryption system that supports threshold decryption, as in [29]. Then the key sk_* is split across all decryption authorities $D_1, \ldots, D_n$ and destroyed. The secret key sk_* will never be reconstituted at a single location. If needed, the key sk_* can be generated in split form by running a distributed protocol between the decryption authorities $D_1, \ldots, D_n$ as in [17, 5].

  The remaining setup steps are as in Protocol 1. In addition, the log server maintains a counter ctr that is initially set to 0 and is incremented after every decryption transaction completes.

- *Normal operation.* The same as in Protocol 1.

- *Investigator queries.* The protocol is similar to the one in Protocol 2 with the hardware module H replaced by the decryption authorities.

  1. The court C receives a signed request for record (i, j, t) from the investigator I. Once the judge approves the request, the court reads the current state of the counter ctr stored at L.
  2. The court C then sends the ciphertext $\mathrm{ct}_C := \mathrm{Enc}\big(\mathrm{pk}_L,\ \mathsf{ctr},\ (i, j, t)\big)$ to the log server L encrypted under the log server's public key pk_L (here the label is set to ctr) and the ciphertext is signed by the court. The log server adds the data to the log. In addition, the court sends all data associated with the request to L, encrypted under pk_L with label ctr and signed by the court.
  3. Next, the court C sends $\big(\mathsf{ctr},\ (i, j, t),\ \mathrm{ct}_{i,j,t},\ \sigma_C\big)$ to a subset of the decryption authorities that is capable of decrypting the ciphertext. Here σ_C is the court's signature on the request. Each authority logs the request with the log server by sending it the message from C encrypted with pk_L and signed by the authority's signing key. Once the log server confirms that ctr is the current value of its counter the authority applies its share of the secret key sk_* to partially decrypt $\mathrm{ct}_{i,j,t}$ with label (i, j, t). It sends the resulting partial decryption to C.

     As before, the label (i, j, t) ensures that the decryption authority is decrypting a ciphertext associated with (i, j, t) and not some other index (i', j', t'). If the label and the ciphertext $\mathrm{ct}_{i,j,t}$ do not match then C will be unable to combine the partial decryptions into the plaintext $x_{i,j,t}$.
  4. The court C combines all the partial decryptions it receives and sends the fully decrypted $\mathrm{ct}_{i,j,t}$ to the investigator. Remark 1 can be used to ensure that the court does not see the plaintext $x_{i,j,t}$ in the clear.

The security properties are similar to those of Protocol 2, but the tamper resistant hardware assumption is replaced by the assumption that it is not possible to corrupt a majority of the decryption authorities.

Observe that if the court's key is completely compromised by an attacker, that attacker can decrypt arbitrary ciphertexts $\mathrm{ct}_{i,j,t}$ of its choice (and this cannot be avoided), but the log will contain a complete record of which ciphertexts were decrypted.

4 Eliminating a single point of trust using uncleared authorities

In this section we address the problem of using decryption authorities $D_1, \ldots, D_n$ that are not cleared to see the investigator's decryption requests. For example, the data sources S_i and other government and non-government organizations could serve as uncleared authorities. They are responsible for making sure that decryption takes place only after court approval and that all information is properly logged, but they are not allowed to see any details of the investigator's query.

While all our constructions up until now used simple public-key cryptography, the need for "blind" decryption authorities requires more sophisticated tools.

Our starting point is an approach similar to Protocol 3 with the important difference that the court "blinds" every decryption request before sending it to the authorities $D_1, \ldots, D_n$. The bulk of this section focuses on how to blind these requests while enabling the authorities to verify (i) court approval and (ii) proper logging.

Oblivious transfer. Our protocols are derived from a cryptographic mechanism called *oblivious transfer* which has a long and extensive history in the cryptographic literature (see e.g. [26, 16, 24, 8, 19, 21]). In simple terms oblivious transfer is a protocol involving two parties called $\mathcal{S}$ for "sender" and $\mathcal{R}$ for "receiver" where:

- the sender's input is a set of s messages $m_1, \ldots, m_s$, and
- the receiver's input is an index $\ell \in \{1, \ldots, s\}$.

$\mathcal{S}$ and $\mathcal{R}$ interact and at the end of the protocol we must have that: (1) the receiver $\mathcal{R}$ learns message number ℓ, namely m_ℓ, and nothing else, (2) the sender $\mathcal{S}$ learns nothing and is therefore oblivious to the value of ℓ. Modern oblivious transfer protocols (e.g. [24, 8, 19, 21]) allow the receiver to request multiple messages and even do so adaptively (i.e. the receiver may choose each query based on previous executions of the protocol).

To see the relation to our problem we can think of the decryption authorities $D_1, \ldots, D_n$ as if they were holding all the plaintexts $\{x_{i,j,t}\}$. We can think of the investigator as having a set of indexes $\mathcal{R} = \{(i, j, t)\}$ of interest and it wishes to retrieve from the decryption authorities all plaintexts associated with indexes in $\mathcal{R}$. The investigator should learn nothing about plaintexts outside of $\mathcal{R}$ and the decryption authorities should learn nothing about the set $\mathcal{R}$.

4.1 Auditable oblivious transfer

While oblivious transfer is closely related to our problem, the current formulation of oblivious transfer does not quite meet our needs. Using a basic oblivious transfer protocol between the investigator and the data sources, we would have no way of ensuring that the records accessed by the investigator were the same ones authorized by the court. Thus, we need a primitive with stronger accountability properties.

In particular, we need to ensure that (1) every decryption transaction is properly logged on the log server, (2) the oblivious sender can be implemented as a set of decryption authorities that must be involved in every transfer, and (3) the data sources S_i do not hold any long-term secrets. We therefore first re-formulate the concept of oblivious transfer to address these issues and then construct our system by adapting a specific oblivious transfer protocol to these settings.

We refer to this concept as *Auditable oblivious transfer* (aOT). In its simplest form an aOT is a two-party functionality between $\mathcal{R}$ and $\mathcal{S}$ defined as follows: a trusted party T generates a key pair $\mathrm{pk}_{\mathrm{L}}, \mathrm{sk}_{\mathrm{L}}$ for a public-key encryption system and a key pair $\mathrm{pk}_{\mathrm{C}}, \mathrm{sk}_{\mathrm{C}}$ for a digital signature scheme. The functionality is then:

- inputs: $\mathcal{R}$ is given $(\mathrm{sk}_{\mathrm{C}},\ \mathrm{pk}_{\mathrm{L}},\ \ell)$ for some $\ell \in I$, and
  $\mathcal{S}$ is given $(\{m_\ell\}_{\ell \in I},\ \mathrm{pk}_{\mathrm{C}},\ \mathrm{pk}_{\mathrm{L}})$.
- outputs: $\mathcal{R}$ obtains m_ℓ, and
  $\mathcal{S}$ obtains an auditing record, namely $\big(c := \mathrm{Enc}(\mathrm{pk}_{\mathrm{L}}, \ell),\ \mathrm{Sign}(\mathrm{sk}_{\mathrm{C}}, c)\big)$.

This captures the fact that $\mathcal{S}$ obtains an auditing record for the request that $\mathcal{R}$ made. This can be extended to multiple (adaptive) message requests by $\mathcal{R}$ where for each request, $\mathcal{S}$ obtains an auditing record. The formulation would mirror the adaptive definition of oblivious transfer as in [8].

Here party $\mathcal{S}$ represents both the decryption authorities and the log server. Party $\mathcal{R}$ represents both the investigator and the court.

This notion still does not fully capture our needs. First, the the sender $\mathcal{S}$ needs to be split into n parties $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and all (or most) need to be contacted for every transfer. Second, in our settings the sender $\mathcal{S}$ does not hold the messages $\{m_\ell\}_{\ell \in I}$. Instead, the receiver $\mathcal{R}$ has encryptions of these messages. This captures the fact that the data sources S_i can only encrypt the data under some public-key, but do not hold long-term secrets.

Auditable threshold oblivious transfer. We further extend the functionality as follows. An auditable threshold oblivious transfer is a tuple of interactive machines $(\mathcal{R}, \mathcal{S}_1, \ldots, \mathcal{S}_n)$. We describe (informally) the operation of these machines as an abstract multi-party functionality. The functionality is parametrized by a public-key encryption scheme and a signature scheme. To describe the functionality we need a bit of setup. First, a trusted party T generates two key pairs $\mathrm{pk}_{\mathrm{L}}, \mathrm{sk}_{\mathrm{L}}$ and $\mathrm{pk}_*, \mathrm{sk}_*$ for the public-key encryption system. It creates shares $(\mathrm{sk}_*^{(1)}, \ldots, \mathrm{sk}_*^{(n)})$ of sk_*. The trusted party also generates a key pair $\mathrm{pk}_{\mathrm{C}}, \mathrm{sk}_{\mathrm{C}}$ for the digital signature scheme. Second, let $M = \{m_\ell\}_{\ell \in I}$ be a set of messages indexed by some set I. The trusted party T encrypts all messages in M under the public-key pk_*, namely $C_* := \{\mathrm{ct}_\ell := \mathrm{Enc}(\mathrm{pk}_*, \ell, m_\ell)\}_{\ell \in I}$. The set C_* corresponds to the set of ciphertexts held by the investigator in the real-world setup.

The functionality computed by $(\mathcal{R}, \mathcal{S}_1, \ldots, \mathcal{S}_n)$ can now be described as follows. The machines are given the following inputs:

- inputs: $\mathcal{R}$ is given $(C_*,\ \mathrm{sk}_{\mathrm{C}},\ \mathrm{pk}_*,\ \mathrm{pk}_{\mathrm{L}},\ \ell)$ for some $\ell \in I$, and
  for $i = 1, \ldots, n$ machine $\mathcal{S}_i$ is given $(\mathrm{sk}_*^{(i)},\ \mathrm{pk}_{\mathrm{C}},\ \mathrm{pk}_{\mathrm{L}})$.

- outputs: $\mathcal{R}$ obtains m_ℓ, and
  for $i = 1, \ldots, n$ machine $\mathcal{S}_i$ obtains an auditing record,
  namely $\big(c := \mathrm{Enc}(\mathrm{pk}_{\mathrm{L}}, 0, \ell),\ \mathrm{Sign}(\mathrm{sk}_{\mathrm{C}}, c)\ \big)$.

Even if all parties $\mathcal{S}_1, \ldots, \mathcal{S}_n$ collude, they learn nothing from the protocol beyond the auditing record. Similarly, even if $\mathcal{R}$ colludes with a sub-threshold coalition of $\mathcal{S}_1, \ldots, \mathcal{S}_n$ they can learn nothing about the decryption of any other ciphertext $\mathrm{ct}_{\ell'} \in C_*$ for $\ell' \neq \ell$.

The goals stated informally above can be extended to multiple (adaptive) message requests by $\mathcal{R}$. After each request, $\mathcal{R}$ learns the requested plaintext and the parties $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$ obtain an auditing record.

These definitions can be made precise using the real-world/ideal-world paradigm along the lines formulated in [8] for adaptive oblivious transfer.

4.2 Cryptographic primitives

We construct an auditable threshold oblivious transfer by adapting an adaptive oblivious transfer protocol due to Green and Hohenberger [19]. This protocol is the best suited for our needs and enables us to satisfy the requirement that the data sources S_i not hold any long-term secrets. Before describing our system we first review the cryptographic primitives from which it is built.

4.2.1 Identity based encryption (IBE)

The oblivious transfer of [19] makes use of identity-based encryption, and in particular the Boneh-Boyen IBE [2]. Recall that identity-based encryption (IBE) [27, 6] is a generalization of public key encryption where public keys can be arbitrary strings. Here we will use data identifiers (i, j, t) as IBE public keys.

Briefly, an IBE system is made up of four algorithms:

$\mathsf{Setup}_{\mathrm{IBE}}$ generates a master public-key mpk and a master secret key msk,

$\mathsf{Extract}_{\mathrm{IBE}}$ uses the master secret key msk to generate a secret key sk for a given public-key (i, j, t),

$\mathsf{Enc}_{\mathrm{IBE}}(\mathrm{mpk}, (i, j, t), m)$ encrypts a message m to a public-key (i, j, t), and

$\mathsf{Dec}_{\mathrm{IBE}}(\mathrm{sk}, c)$ decrypts a ciphertext c with a secret key sk.

The security requirement is that an adversary who obtains the secret keys for multiple identities cannot break the security of ciphertexts encrypted for some other identity.

4.2.2 The Boneh-Boyen IBE system (BB-IBE)

Many IBE systems in the literature make use of pairings (e.g. [5, 2, 30, 18]). A pairing is an efficiently computable (non-trivial) bilinear mapping $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ where $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are finite cyclic groups of some prime order p. We let g be a generator of $\mathbb{G}$ and $\hat{g}$ be a generator of $\hat{\mathbb{G}}$. With this setup the BB-IBE works as follows [2]:

- $\mathsf{Setup}_{\mathrm{IBE}}$ chooses a random α in $\mathbb{Z}_p$, random g_1, h in $\mathbb{G}$, and consistent random $\hat{g}_1, \hat{h}$ in $\hat{\mathbb{G}}$ as described in [2]. It computes $v = e(g_1, \hat{g})^\alpha$ and outputs

$$\mathrm{mpk} = (g, g_1, h, \hat{g}, \hat{g}_1, \hat{h}, v) \quad \text{and} \quad \mathrm{msk} = g_1^\alpha$$

- $\mathsf{Extract}_{\mathrm{IBE}}(\mathrm{msk}, \mathsf{id})$ chooses a random s in $\mathbb{Z}_p$ and outputs $\mathrm{sk}_{\mathsf{id}} := \big(\ \mathrm{msk} \cdot (g_1^{\mathsf{id}}\ h)^s,\ \ g^s\big)$.

- $\mathsf{Enc}_{\text{IBE}}(\text{mpk}, \mathsf{id}, m)$ chooses a random r in $\mathbb{Z}_p$ and outputs $\text{ct} := \big(\ m \cdot v^r, \ \ \hat{g}^r, \ \ (\hat{g}_1^{\mathsf{id}} \ \hat{h})^r \ \big)$.

There is a similar decryption algorithm $\mathsf{Dec}_{\text{IBE}}(\text{sk}_{\mathsf{id}}, \text{ct})$, but its exact operation is not relevant to our discussion here. Selective IBE security follows from a standard assumption on pairing friendly elliptic curves. Note that encryption does not use pairings and is comparable in speed to a standard elliptic-curve public-key encryption.

4.2.3 Blind-IBE

The adaptive oblivious transfer of [19] exploits a property of BB-IBE called *blind IBE*, a concept introduced in [19]. At a high level, a blind-IBE is an IBE system that supports "blind extraction," namely it is possible to request from the extract algorithm a secret key sk for a public key id without revealing to the algorithm the value of id. The extract algorithm is effectively "blind" to the value of id. We refer to the discussion in [19] for a complete definition of blind-IBE and its security definition.

We let $\mathsf{BlindExtract}_{\text{IBE}}$ denote the two-party protocol to blindly generate a secret key for identity id. Protocol $\mathsf{BlindExtract}_{\text{IBE}}(\mathsf{id})$ runs between one party $\mathcal{R}$ who knows id and another party $\mathcal{S}$ who knows msk. Both have mpk.

The detailed blind extract protocol $\mathsf{BlindExtract}_{\text{IBE}}$ for BB-IBE can be found in [19, Fig. 1]. Here we sketch the main idea.

1. Party $\mathcal{R}$ chooses a random y in $\mathbb{Z}_p$ and computes

$$h' \leftarrow g_1^{\mathsf{id}} \, g^y \ . \tag{1}$$

   Observe that this h' reveals no information about id. Party $\mathcal{R}$ sends h' to $\mathcal{S}$ and in addition proves in zero-knowledge to $\mathcal{S}$ that it knows (y, id) such that $h' = g^y \, g_1^{\mathsf{id}}$.

2. Party $\mathcal{S}$ chooses a random s in $\mathbb{Z}_p$ and computes $\text{sk}'_{\mathsf{id}} := \big(\ \text{msk} \cdot (h'h)^s, \ \ g^s \big)$. It sends sk'_{id} back to $\mathcal{R}$.

3. Party $\mathcal{R}$ checks that the blinded key $\text{sk}'_{\mathsf{id}} = (d'_0, d'_1)$ is well formed by checking that the following equality holds:

$$v \cdot e(d'_1, \hat{h}\hat{h}') = e(d'_0, \hat{g}) \ . \tag{2}$$

   If not, the received key is invalid and $\mathcal{R}$ aborts. If equality holds then $\mathcal{R}$ unblinds the key and re-randomizes it by choosing a random z in $\mathbb{Z}_p$ and computing

$$\text{sk}_{\mathsf{id}} = \big(d'_0 \cdot (g_1^{\mathsf{id}} \ h)^z / d_1'^y, \ \ d'_1 g^z \big)$$

The resulting key sk_{id} is a valid key for the identity id, but $\mathcal{S}$ learns nothing about id. Similarly, $\mathcal{R}$ learns nothing about the secret keys of other identities. The complete proof of security for this protocol is provided in [19].

4.2.4 Secret sharing the IBE master secret

In our settings the master secret key msk is split across a number of decryption authorities. All (or most) of the decryption authorities are needed to generate the secret key for a given identity id. The number of decryption authorities and the threshold needed to generate a secret key is a matter of policy.

Splitting up the master secret in the protocol $\mathsf{BlindExtract}_{\text{IBE}}$ from Section 4.2.3 is done as follows:

1. *Shared IBE setup:* The n parties $\mathcal{S}_1, \ldots, \mathcal{S}_n$ generate a sharing of the secret key msk. That is, party $\mathcal{S}_i$ obtains a share msk_i where $\text{msk}_i = g_1^{\alpha_i}$ and where $(\alpha_1, \ldots, \alpha_n)$ is a linear sharing of α in $\mathbb{Z}_p$. Each party $\mathcal{S}_i$ publishes $v_i = e(g_1, \hat{g})^{\alpha_i}$. The global mpk value $v = e(g_1, \hat{g})^{\alpha}$ is computed and published as before. Standard protocols (e.g. [17]) enable the n parties to generate this shared msk and publish the corresponding $v, v_1, \ldots, v_n$.

2. *Distributed blind key extraction:* Distributed blind extraction for $\mathsf{sk}_{\mathsf{id}}$ is similar to the non-distributed protocol. Steps (1) and (2) of the protocol $\mathsf{BlindExtract}_{\text{IBE}}$ in Section 4.2.3 remain unchanged, except that $\mathcal{S}$ sends h' to a set of active parties $W \subseteq \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ and they all respond as in step (2) in Section 4.2.3. Thus, party $\mathcal{R}$ receives

$$\mathrm{sk}'_{\mathsf{id},i} := \big(\ \mathrm{msk}_i \cdot (h'\, h)^{s_i},\ \ g^{s_i}\big) \qquad \text{for } i \in W\ . \tag{3}$$

For $i \in W$ party $\mathcal{R}$ checks that $\mathrm{sk}'_{\mathsf{id},i} = (d'_{0,i}, d'_{1,i})$ is well formed using Eq. (2), but using v_i instead of v. If any of the checks fail party $\mathcal{R}$ aborts. Otherwise, $\mathcal{R}$ combines all the $\mathrm{sk}'_{\mathsf{id},i}$ and re-randomizes the result by choosing a random z in $\mathbb{Z}_p$ and computing

$$\mathrm{sk}_{\mathsf{id}} = \left(\prod_{i \in W} (d'_{0,i}/d'^{y}_{1,i})^{\lambda_i} \cdot (g_1^{\mathsf{id}}\, h)^z,\quad \prod_{i \in W} (d'_{1,i})^{\lambda_i} g^z \right) \tag{4}$$

where $\{\lambda_i\}_{i \in W}$ are the linear secret sharing coefficients, namely $\alpha = \sum_{i \in W} \lambda_i \alpha_i$.

Proving that an unauthorized set of decryption authorities cannot generate secret keys is a standard argument on information theoretic linear secret sharing schemes (e.g., Shamir secret sharing).

4.2.5 Chosen ciphertext-secure public-key encryption from IBE

The final tool we need is a chosen ciphertext-secure system built from IBE. Canneti, Halevi, and Katz [10, 4] showed how to construct a chosen-ciphertext secure public-key encryption system from any (selective) secure IBE. Their construction works as follows:

- KeyGen: runs $\mathsf{Setup}_{\text{IBE}}$ and outputs mpk as the public key and msk as the secret key.
- Enc(mpk, label, m): generates a fresh signing and verification key pair (sk, vk) for a (one-time) signature scheme, computes $\mathrm{ct}_{\text{IBE}} := \mathsf{Enc}_{\text{IBE}}\big(\mathrm{mpk},\ (\mathsf{label}, \mathrm{vk}),\ m\big)$, and outputs

$$\mathrm{ct} := \big(\ \mathrm{ct}_{\text{IBE}},\ \mathrm{vk},\ \ \mathsf{sig}\ \big) \tag{5}$$

  where sig is a signature on the first two ciphertext components, generated using sk. Note that $(\mathsf{label}, \mathrm{vk})$ is used as the IBE identity to encrypt m.

- Dec(msk, label, ct): parse ct as $(\mathrm{ct}_{\text{IBE}}, \mathrm{vk}, \mathsf{sig})$. Check that sig is a valid signature on the first two components under vk and output $\perp$ if not. If sig is valid, use msk to generate the IBE secret sk_{IBE} key for identity $(\mathsf{label}, \mathrm{vk})$ and output $\mathsf{Dec}_{\text{IBE}}(\mathrm{sk}_{\text{IBE}}, \mathrm{ct}_{\text{IBE}})$.

When applying this to the BB-IBE we need a collision resistant hash function $H_1 : \{0,1\}^* \to \mathbb{Z}_p$. We obtain a chosen-ciphertext secure public-key encryption system where the encryption algorithm Enc(mpk, label, m) works as follows: first, it chooses a random r in $\mathbb{Z}_p$ and generates a fresh signing and verification key pair (sk, vk) for a (one-time) signature scheme. Next, it computes $w := (g_1^{H_1(\langle \mathsf{label}, \mathrm{vk} \rangle)}\, h)$ and outputs

$$\mathrm{ct} := \big(\ v^m \cdot v^r,\ \ g^r,\ \ w^r,\ \ \mathrm{vk},\ \ \mathsf{sig}\big) \tag{6}$$

where sig is as in (5), namely a signature on the left four elements using key sk (for convenience we swapped the roles of the groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ so that the ciphertext lives in $\mathbb{G}$).

Note that the message m in the ciphertext (6) is encoded as $v^m \in \mathbb{G}_T$ which will be convenient later when we use this system. This encoding means that the decryption algorithm outputs v^m instead of m. Recovering m will require computing the discrete-log of v^m base v. Fortunately this is not a problem because when we use this system the message m will always be short, say 64-bits, so that computing discrete log can be done in time about 2^{32} using Pollard's Kangaroo method [25]. This should only take a few minutes on a modern computer.

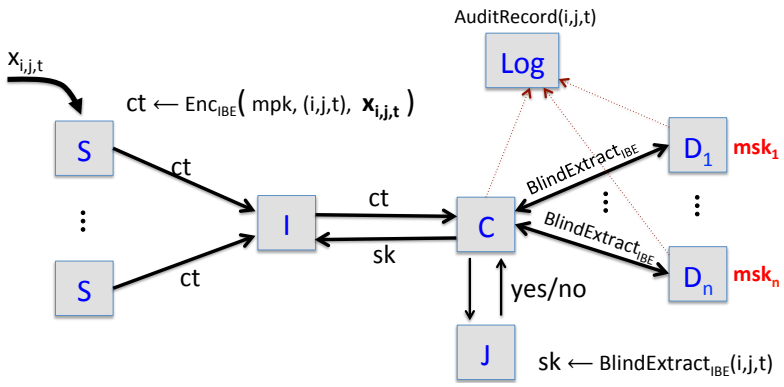

Figure 3: Using uncleared decryption authorities. The parties involved are as in Figure 2 except that now all requests to the decryption authorities are blinded as in Eq. (1). As before the authorities are needed for every decryption request and they help ensure that every request is properly logged at L.

4.3 The complete system

Using the building blocks described above, we can now describe the complete system. We describe the system in terms of the usual parties, namely S_i, A, C, L, D_j (see Figure 3). The underlying auditable threshold oblivious transfer system is obtained by merging the investigator and the court into one entity $\mathcal{R}$, and merging the log server into each D_i to obtain the distributed senders $\mathcal{S}_i$.

Protocol 4 (using uncleared decryption authorities with after-the-fact auditability). The system works as follows:

- *Setup.* the system is initialized as follows:

  1. First, generate a master secret key msk and public parameters mpk in the BB-IBE scheme by running algorithm $\mathsf{Setup}_{\text{IBE}}$. The key msk is split across the n decryption authorities $D_1, \ldots, D_n$ and is destroyed. For $i = 1, \ldots, n$ each D_i now has a share msk_i of msk. It publishes v_i as in Section 4.2.4 along with a (random oracle) proof of knowledge of the corresponding msk_i.

     The master key msk will never be reconstituted at a single location. If needed, msk can be generated in split form by running a protocol from [17] among the decryption authorities $D_1, \ldots, D_n$.
  2. Generate an encryption/decryption key-pair $\text{sk}_{\text{L}}, \text{pk}_{\text{L}}$ for a public-key encryption system. The key sk_{L} will be stored at the logging server and pk_{L} will be published to all parties.
  3. The court C and each decryption authority D_i will generate a signing and verification key pair for a signature scheme and publish the verification keys to all parties.
  4. As in Protocol 3, the log server maintains a counter ctr that is initially set to 0 and is incremented after every decryption transaction completes.

- *Ordinary operation.* Using the setup above the protocol protects data elements as follows:

  1. At the end of each time interval t, data source i computes

     $$\text{ct}_1 := \mathsf{Enc}_{\text{IBE}}(\text{mpk},\ (i,j,t),\ r_{i,j,t})\ , \quad \text{ct}_2 := x_{i,j,t} \oplus H(r_{i,j,t}) \tag{7}$$

     where $r_{i,j,t}$ is a random element in $\mathbb{G}_T$ chosen by the data source S_i, and $H : \mathbb{G}_T \to \mathcal{M}$ is a public hash function modeled as a random oracle.

     The hybrid encryption method in (7) of data $x_{i,j,t}$ using tag (i,j,t) is taken directly from the adaptive oblivious transfer protocol of [19] (presented in their Figure 3).

2. Data source S_i sends $\mathrm{ct}_{i,j,t} = (\mathrm{ct}_1, \mathrm{ct}_2)$ to the investigator I.

- *Investigator queries.* Over a secure channel (e.g. in person, in a courtroom), the investigator I presents to the court C a record tag $\mathsf{id} = (i, j, t)$ and asks for the corresponding ciphertext $\mathrm{ct}_{\mathsf{id}}$ to be decrypted. The investigator I provides its signature on the request. Once the judge approves the request, the court C issues a legal warrant and performs the following operations:

  1. The court C reads the current state of the counter ctr stored at the log server L.
  2. The court C then sends the ciphertext $\mathrm{ct}_{\mathrm{C}} := \mathrm{Enc}\big(\mathrm{pk}_{\mathrm{L}},\ \mathsf{ctr},\ \mathsf{id}\big)$ to the log server encrypted under the log server's public key pk_{L} (here the label is set to ctr) and the ciphertext is signed by the court. The log server adds the data to the log. We denote by $\mathsf{rand}_{\mathrm{C}}$ the randomness used in creating the ciphertext ct_{C}.

     The court can send another ciphertext to the log server L, encrypted under pk_{L} with label ctr and signed by the court, containing all auxiliary information associated with the request, including I's signature on the request. The log server adds the data to the log.
  3. Next, the court C and active decryption authorities $D_1, \ldots, D_n$ engage in protocol $\mathsf{BlindExtract}_{\mathrm{IBE}}$ from Section 4.2.4 to blindly extract $\mathrm{sk}_{\mathsf{id}}$ where C plays the role of $\mathcal{R}$.

     We let y denote the random blinding value chosen by C. The court C computes $h' \leftarrow g_1^{\mathsf{id}}\, g^y$ and then executes the following protocol with each active authority D_i:
     (a) C sends $(h',\ \mathsf{ctr})$ to D_i.
     (b) The authority D_i logs the request with the log server by sending it the message from C signed by the authority's signing key. The log server responds with ct_{C} and the court's signature on ct_{C}, and confirms that ctr is the current value of its counter ctr. If not or if the court's signature on ct_{C} is invalid, the decryption authority aborts.
     (c) Next, C proves in zero knowledge to D_i that it knows (y, id) such that $h' = g_1^{\mathsf{id}}\, g^y$.
     If the proof fails the authority aborts.
     (d) Finally, authority D_i uses its share msk_i of the master secret msk to derive $\mathrm{sk}'_{\mathsf{id},i}$ as in Eq. (3) and sends this value back to C.
  4. The court C verifies the received blinded values $\mathrm{sk}'_{\mathsf{id},i}$ as in Section 4.2.4 (using Eq. (2)) and combines all the partial keys $\mathrm{sk}'_{\mathsf{id},i}$ using Eq. (4) to obtain the (unblinded) IBE decryption key $\mathrm{sk}_{\mathsf{id}}$ needed to decrypt $\mathrm{ct}_{\mathsf{id}}$. It sends $\mathrm{sk}_{\mathsf{id}}$ to the investigator I over a secure channel.
  5. The investigator I receives $\mathrm{sk}_{\mathsf{id}}$ and decrypts the record $\mathrm{ct}_{\mathsf{id}}$ by invoking $\mathsf{Dec}_{\mathrm{IBE}}(\mathrm{sk}_{\mathsf{id}},\ \mathrm{ct}_{\mathsf{id}})$.

Security of the system. The underlying adaptive oblivious transfer protocol was shown to be fully simulation secure by Green and Hohenberger [19, Thm. 4.4]. Informally, this proves that (i) the parties $L, D_1, \ldots, D_n$ learn nothing beyond the encryption of id under pk_{L}, and (ii) the investigator cannot decrypt any ciphertext $\mathrm{ct}_{\mathsf{id}'}$ for $\mathsf{id}' \neq \mathsf{id}$.

Because every key extraction request is logged, misbehaving parties in this system will be discovered during an after-the-fact audit. However, during the audit it may not be possible to determine which records they accessed. By enhancing the proof of knowledge in Step (3c) to also verify accuracy of the ciphertext ct_{C} written to the log, we can ensure that an after-the-fact audit will reveal which records were accessed by misbehaving parties. This brings us to our final protocol.

Protocol 5 (using uncleared decryption authorities with online auditability). The system works very similarly to Protocol 4.

- *Setup.* The same as Protocol 4.
- *Ordinary operation.* The same as Protocol 4, except that instead of using a standard RSA hybrid public key encryption scheme to encrypt under the escrow key pk_{L}, all of the parties instead use the more advanced IBE-based system of Canetti, Halevi, and Katz, described in Section 4.2.5.

- *Investigator queries.* The same as Protocol 4, except in Step 3(c), the court C proves in zero knowledge to D_i that it knows $(y, \mathsf{id}, \mathsf{rand}_{\mathrm{C}})$ such that

$$h' = g_1^{\mathsf{id}}\, g^y \quad \text{and} \quad \mathrm{ct}_{\mathrm{C}} = \mathrm{Enc}\big(\mathrm{pk}_{\mathrm{L}},\ \mathsf{ctr},\ \mathsf{id}\ ;\ \mathsf{rand}_{\mathrm{C}}\ \big)\ . \tag{8}$$

  Here $\mathsf{rand}_{\mathrm{C}}$ is the randomness used in creating the ciphertext ct_{C} sent to the log server L.

  If the proof fails the authority aborts. This proof of knowledge proves to D_i that h' is a blinding of the identity id contained in ct_{C}. In other words, ct_{C} is an encryption of the identity whose key is being retrieved.

We remark that analogous accountability procedures could be used to include the investigator's signed encrypted requests in the log also, along with the court's signed encrypted orders, and prove consistency between the court's and the investigator's entries, but we omit these procedures here for clarity of exposition.

The efficiency of Step (3c) in Protocol 5. The public key system used by the log server L should be chosen so as to make the proof of knowledge (8) efficient. While any zero knowledge proof of knowledge system can be used we briefly describe one efficient instantiation. Suppose the log server L chooses pk_{L} as a public-key in the public key system described in Section 4.2.5. Then ct_{C} is as follows:

$$\mathrm{ct}_{\mathrm{C}} = \big(\ v_{\mathrm{L}}^{\mathsf{id}} \cdot v_{\mathrm{L}}^r,\quad g^r,\quad w^r,\quad \mathrm{vk},\quad \mathsf{sig}\big) \tag{9}$$

where $w := (g_1^{H_1(\langle \mathsf{ctr}, \mathrm{vk}\rangle)}\, h)$ and $\mathsf{id} \in \mathbb{Z}_p$ is an encoding of (i, j, t). All other elements are as in Section 4.2.5. Parse ct_{C} as $\mathrm{ct}_{\mathrm{C}} = (\mathrm{ct}_0, \mathrm{ct}_1, \mathrm{ct}_2, \mathrm{vk}, \mathsf{sig})$. Then the zero knowledge proof of knowledge in (8) can be done as follows:

1. D_i checks that sig is a valid signature on $(\mathrm{ct}_0, \mathrm{ct}_1, \mathrm{ct}_2, \mathrm{vk})$ under key vk and aborts if not.
2. The court C proves in zero knowledge to D_i that it knows (y, id, r) such that

$$h' = g_1^{\mathsf{id}}\, g^y \quad,\quad \mathrm{ct}_0 = v_{\mathrm{L}}^{\mathsf{id}}\, v_{\mathrm{L}}^r \quad,\quad \mathrm{ct}_1 = g^r \quad,\quad \mathrm{ct}_2 = w^r \quad,\quad 0 \le \mathsf{id} < 2^{64}\ . \tag{10}$$

   Proving knowledge of such (y, id, r) can be done efficiently using standard zero knowledge techniques [13]. The range proof on id (the right most statement in (10)) is needed to ensure that the discrete-log of $v_{\mathrm{L}}^{\mathsf{id}}$ base v_{L} can be computed efficiently during decryption. It can be done efficiently as in [12, 7, 11]. Note that 64 bits is more than enough to encode $\mathsf{id} = (i, j, t)$.

   In some cases it may be desirable to embed more information in the identifier id, in addition to (i, j, t), causing the identifier id to be longer than 64 bits. This can be accommodated by breaking the identifier into blocks of 64 bits and encrypting all blocks as in (9), all with the same label ctr. The proof of knowledge (10) will need to be augmented with a simple consistency check to prove that the sum of all the blocks is equal to the value of id used in the definition of h'.

5 Prototype implementation

To assess the feasibility of our protocols, we have an implementation in progress for Protocol 4. The prototype code includes an implementation of all of the necessary cryptographic primitives, among them a complete and fully-functional implementation of the Boneh-Boyen identity-based encryption (BB-IBE) scheme [2].

In Protocol 4, by far the most computationally-demanding step is the step in which the data sources encrypt the data records to be sent to the data custodian. As noted in Section 1.2, these records may include one data record, possibly an empty one, for every possible user j during each time interval t, to prevent attacks based on traffic analysis and data volume. This amounts to a potentially large volume of data that must be processed during each time interval t (in our example, each day). If we are dealing with a telecommunications carrier or a large cloud service provider, the number of such data records might range

in the tens to hundreds of millions per day. For our proposed protocol to be viable, the data sources must be able to process these all of these data in a timely manner each day.

The steps involved for the data sources to process these data records are as follows. Recall from Section 4.3 that the data sources in Protocol 4 uses hybrid encryption: that is, in order to encrypt a data record $x_{i,j,t}$, they first encrypt the record using a symmetric scheme that provides authenticated encryption, and then encrypt the underlying symmetric key using the IBE scheme under the identity derived from (i, j, t) (in our implementation, we apply SHA-256 to a safely-encoded form of the tuple (i, j, t), and use the output as the identity in the IBE scheme).

In our benchmarks, we measure the time necessary for the data sources to process 500 million data records in this manner. This number is intended to estimate the total number of records that might be generated across multiple telecommunication carriers in a single day, which is a plausible duration for the time interval t in our model. Each data record consists of a header (i, j, t) and up to 1 KB worth of data, which could represent daily email or phone call metadata in a telecommunications database.

Our implementation code uses 128-bit AES/GCM for its authenticated encryption scheme, as well as the BB-IBE scheme over a 289-bit MNT elliptic curve. Both of these choices target a 128-bit security level, as 128-bit security is the widely adopted cryptographic standard with ubiquitous library support. (We remark, however, that it would be straightforward to extend the protocol to a 256-bit security level for real-world deployments, and our benchmarks indicate that this would only slow down the main computational bottleneck by about a factor of two.) The code for our prototype in progress includes a complete implementation of the BB-IBE cryptosystem written in C. We use the PBC library [23] for elliptic-curve and pairing support, and the GNU Multiple Precision Arithmetic Library (GMP) for big-integer arithmetic. We also use the OpenSSL crypto library for some additional cryptographic primitives such as AES/GCM and SHA-256.

In our implementation benchmark, we encrypt each record first using 128-bit AES/GCM and then encrypt the AES key using BB-IBE under the identity derived from (i, j, t). Since this process is largely parallelizable, we distribute the work across a cluster with 500 cores (by current standards, this would roughly correspond to a rack of machines). More concretely, each node in the cluster is responsible for processing a subset of the records. Our results demonstrate that using a small computing cluster of 500 cores, we can process 500 million records in just under two hours.

We also give an estimate of the amount of time needed to process a warrant request via the procedure described in Protocol 4. Using our BB-IBE implementation, we measured the amount of time needed for all IBE operations in the protocol. With 10 decryption authorities, the total time needed for IBE operations across all parties is just under 3 seconds. The remaining pieces of the warrant request component of Protocol 4 consist exclusively of public-key operations (encryption and signing) as well as network communication over TLS between the different parties. With 10 decryption authorities, a conservative bound on the total number of public key operations is 100, and if we use 2048-bit RSA (targeting a 128-bit security level) for the public key operations, all of these operations can be completed in under a minute. Factoring in network latencies and the cost of communication over TLS, a conservative estimate for the time it takes for a complete execution of the warrant request protocol is under 5 minutes. With this analysis and the above benchmarks, we conclude that Protocol 4 is feasible at scale on large data sets consisting of billions of records.

6 Extensions

Our system can be extended in several ways. We list a few directions here.

Storing ciphertexts at a third party (the "data custodian" U). As mentioned in Section 2, for clarity we have described all of our protocols in terms of direct communication of ciphertexts between data sources and the investigator. However, such a design choice would sacrifice the vital "Shannon" security property (Section 1.3): if the investigator can defeat the cryptosystem used to encrypt the records, or otherwise obtain the master secret key, then it might be able to access many records without getting an order or transmitting any additional data over the network—potentially making the system more vulnerable to abuse in practice than standard solutions without cryptographic protocols. Thus, we do not believe that this kind of direct

message flow is viable in a real-world deployment. Instead, we now describe in more detail how to store the ciphertexts at a third party, the "data custodian" U, who can serve one ciphertext to the investigator for each valid (signed) order.

In a simple version of this scheme, the data source will tag each encrypted record with a tag $P(K_T, (i, j, t))$ where P is a pseudorandom permutation and K_T is a symmetric key known to the data source and the investigator. When an order is issued for identifier (i', j', t'), the investigator will ask the intermediary for the record tagged $P(K_T, (i', j', t'))$.

Although the data custodian does not itself verify that the encrypted record it produces is the one covered by the order, it does ensure the Shannon property that the number of encrypted records received by the investigator does not exceed the number of orders that have been issued. Because the investigator receives only one encrypted record per order, the worst it can do by breaking the encryption on records is to get an order for one record and then access some other (single) record.

The data source does not learn which record was fetched, because only the investigator and the intermediary see the permuted tag value that is accessed. The intermediary does not learn which underlying record was accessed because it sees only the permuted tag. However, if the data source and the intermediary collude, they can learn which record was accessed. If necessary, this collusion can be addressed by using a longer chain of permutations controlled by multiple parties.

Refined access policies among decryption authorities. In Sections 3 and 4 we used multiple decryption authorities $D_1, \ldots, D_n$ to decrypt a target ciphertext. For reasons of availability one may want to only require $t < n$ authorities to be contacted for every transaction. The classic Shamir secret sharing approach lets us designate any values for t and n. The exact settings for t and n are a matter of policy.

For a system like ours, however, we may not want *any* set of t authorities to enable decryption. For example, if the court C runs a decryption authority we may require that authority to participate in every decryption request. This is easily done by additively splitting the secret key α as $\alpha = \alpha_1 + \alpha_2$ in $\mathbb{Z}_p$ for random α_1 and α_2. The share α_1 is given to the court C and the share α_2 is split via Shamir secret sharing among the remaining authorities. This way the court must participate in every decryption request and any additional t of the remaining authorities can be used to complete the request. Generally, any monotone access structure recognizable by a poly-size formula can be implemented using a linear secret sharing scheme [1].

Using a mixture of cleared and uncleared decryption authorities. Our system in Section 4 was described with uncleared decryption authorities in mind who are not authorized to view the contents of warrants. The messages sent to all authorities were blinded so as not to reveal any information about the warrant. When using a mix of decryption authorities, some cleared and some uncleared, the court need not use the blinding mechanism with the cleared authorities. The warrant subject (i, j, t) along with the randomness rand_C used to create the ciphertext ct_C can be sent unblinded to the cleared authorities. These authorities will retrieve ct_C from the log server, verify its authenticity, and if authentic send back their share of the secret key $\mathrm{sk}_{i,j,t}$. This simplifies the protocol between the court C and the cleared authorities.

Restricting access to sub-records. As we described the system in Section 1.2 data source S_i collects data records about user j during time period t, say one day, and this aggregate set of records forms the plaintext data element $x_{i,j,t}$ identified by (i, j, t). This $x_{i,j,t}$ is encrypted to form the ciphertext $\mathrm{ct}_{i,j,t}$. When the investigator is given the ability to decrypt $\mathrm{ct}_{i,j,t}$ it obtains access to *all* of $x_{i,j,t}$. In some cases it may be desirable to give the investigator access to only a single record inside of $x_{i,j,t}$.

Clearly one option is to shrink the time window from one day to, say, one minute so that $x_{i,j,t}$ only holds records collected during a one minute window. The problem is that S_i will likely need to create a record $x_{i,j,t}$ even if user j had no activity during time period t. Otherwise, the absence of a ciphertext $\mathrm{ct}_{i,j,t}$ for time t indicates that user j was inactive during time t and it may be desirable to not leak this information. Consequently, if dummy records are used then shrinking the time window to one minute is undesirable since it may introduce many dummy records and increase storage costs. A larger time window, say one day, greatly

reduces the number of dummy records in the system, but suffers from revealing *all* of user j's activity during time t, even if a warrant applies to only one record during that period.

When using our third scheme a simple solution is to encrypt every record under the IBE public key (i, j, T) where T is the precise time that the associated event took place. The set of resulting ciphertexts for events that took place during time period t is assembled to form $x_{i,j,t}$, possibly padding $x_{i,j,t}$ to some minimum length. This container $x_{i,j,t}$ is then encrypted under the IBE public key (i, j, t) to form $\mathrm{ct}_{i,j,t}$. When a warrant is issued for an event at time T the decryption authorities first release the secret key $\mathrm{sk}_{i,j,t}$ to enable decryption of $\mathrm{ct}_{i,j,t}$. Then another round with the decryption authorities releases the desired secret key $\mathrm{sk}_{i,j,T}$ that provides access to the record at time T and nothing else. This approach prevents leaking the amount of activity for each user in the clear without adding too much storage overhead.

7 Conclusion

In this work, we describe a series of cryptographic protocols to facilitate the secure execution of warrants and legal orders authorizing access to data held by private parties. There are many natural scenarios where such a protocol might be useful, for instance, in the scenario where law enforcement agencies need to obtain legal clearance in order to access data about individuals. We present five protocols that achieve different security guarantees and require varying degrees of complexity.

In our first and second protocols, we require that the court maintain a secret key used to control access to plaintexts. While conceptually very simple, this protocol has the undesirable property that there is a single point of failure, namely, if the court's secret key is compromised, the adversary can potentially obtain access to arbitrary records. To remove this single point of failure, in protocol three, we present a protocol that distributes the record decryption process across multiple parties. The drawback in this protocol, however, was that the decryption authorities learned the investigator's request. To address this potential problem, we present two additional protocols based on a small extension to adaptive oblivious transfer. In these protocols, the decryption authorities do not learn the investigator's query; thus, the protocols satisfy a stronger notion of security.

To assess the feasibility of our proposed protocols, we benchmarked some of the core cryptographic primitives in Protocol 4 (which satisfies a reasonably strong notion of security and provides after-the-fact auditability). Our benchmarks indicate that the most computationally-intensive component of the protocol, that is the process of encrypting all of the records, can be performed efficiently. Specifically, processing a database of 500 million records takes under two hours on a small computing cluster. Since this process is entirely parallelizable, scaling up to handling billions of records is not a problem. Moreover, our benchmarks for the underlying cryptographic primitives indicate that a warrant request can be processed in under five minutes. These experiments thus demonstrate the viability of our protocol as a means for secure execution of warrants.

By enabling more secure execution of warrants, our protocols create new public policy options. For example, in policy settings where the contents of orders are secret but accountability is especially important, such as warrants for access to communications data for foreign intelligence, our protocols show that accountability can be increased while at the same time maintaining or improving the security of the warrant process and protecting against abuse or the failure of technical protections. More generally, methods like ours create the hope that the execution of orders and warrants can be more secure for honest investigators, more protective of privacy for those who are not the subject of legitimate orders, and subject to more effective oversight by political institutions and the public.

Acknowlegments

The work was supported by the National Science Foundation.

References

[1] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Technion Institute of Technology, Haifa, Israel, 1996.

[2] Dan Boneh and Xavier Boyen. Efficient selective identity-based encryption without random oracles. *J. Cryptology*, 24(4):659–693, 2011. extended abstract in Eurocrypt 2004.

[3] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA*, pages 226–243, 2006.

[4] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.

[5] Dan Boneh and Matthew K. Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, 2001.

[6] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003. Preliminary version in *Advances in Cryptology – CRYPTO '01*, pages 213–229, 2001.

[7] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.

[8] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT*, pages 573–590, 2007.

[9] Ran Canetti and Shafi Goldwasser. An efficient *threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT*, pages 90–106, 1999.

[10] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.

[11] Rafik Chaabouni, Helger Lipmaa, and Abhi Shelat. Additive combinatorics and discrete logarithm based range protocols. In *ACISP*, pages 336–351, 2010.

[12] Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In *EUROCRYPT*, pages 561–575, 1998.

[13] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[14] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, pages 307–315, 1989.

[15] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[16] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[17] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[18] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

[19] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *ASIACRYPT*, pages 265–282, 2007.

[20] IBM. Cryptocards: IBM systems cryptographic hardware products. www-03.ibm.com/security/cryptocards/.

[21] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, pages 577–594, 2009.

[22] Seny Kamara. Are compliance and privacy always at odds? Available at http:/outsourcedbits.org//2013/07/23/are-compliance-and-privacy-always-at-odds/, July 2013.

[23] Ben Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, 2009. http://crypto.stanford.edu/pbc/thesis.pdf.

[24] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *CRYPTO*, pages 573–590, 1999.

[25] John Pollard. Kangaroos, monopoly and discrete logarithms. *J. Cryptology*, 13(4):437–447, 2000.

[26] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981.

[27] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – CRYPTO '84*, pages 47–53, 1984.

[28] Victor Shoup. Encryption algorithms – part 2: Asymmetric ciphers. ISO Standard 18033-2, May 2006. www.shoup.net/iso/.

[29] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002.

[30] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.