OBLIVIOUS ENFORCEMENT OF HIDDEN INFORMATION RELEASE POLICIES USING ONLINE CERTIFICATION AUTHORITIES

by

Brian Wongchaowart

B.S. in Computer Science,

University of Pittsburgh, 2008

Submitted to the Graduate Faculty of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2010

UNIVERSITY OF PITTSBURGH

ARTS AND SCIENCES

This thesis was presented

by

Brian Wongchaowart

It was defended on

August 5, 2010

and approved by

Adam J. Lee, Assistant Professor, Department of Computer Science

Panos K. Chrysanthis, Professor, Department of Computer Science

Alexandros Labrinidis, Associate Professor, Department of Computer Science

Thesis Advisor: Adam J. Lee, Assistant Professor, Department of Computer Science

OBLIVIOUS ENFORCEMENT OF HIDDEN INFORMATION RELEASE POLICIES USING ONLINE CERTIFICATION AUTHORITIES

Brian Wongchaowart, M.S.

University of Pittsburgh, 2010

This thesis examines a new approach to attribute-based access control with hidden policies and hidden credentials. In this setting, a resource owner has an access control policy that is a function of Boolean-valued attributes of the resource requester. Access to the resource should be granted if and only if the resource owner's policy is satisfied, but we wish to hide the access control policy from the resource requester and the requester's attributes from the resource owner.

Previous solutions to this problem involved the use of cryptographic credentials held by the resource requester, but it is obvious that if no information is provided about the access control policy, then the resource requester must try to satisfy the policy using every available credential. An initial contribution of this thesis is the first published empirical evaluation of the state-of-the-art protocol of Frikken, Atallah, and Li for access control with hidden policies and hidden credentials, demonstrating that the computational cost of the required cryptographic operations is highly burdensome.

A new system model is then proposed that includes the active involvement of online certification authorities (CAs). These are entities that can provide authoritative information about the attributes in a resource owner's access control policy. Allowing the resource owner to query these online CAs immediately removes the need for the resource requester to guess which credentials to use.

If the resource owner was allowed to learn the values of a requester's attributes from online CAs, however, the requester's credentials would no longer be private. This thesis examines cryptographic solutions in which the CAs' replies do not directly reveal any attribute information to the resource owner, but can nevertheless be used in the enforcement of an access control policy. The techniques considered involve scrambled circuit evaluation, homomorphic encryption, and secure multiparty computation using arithmetic circuits and Shamir secret sharing. Empirical experiments demonstrate that the proposed protocols can provide an order-of-magnitude performance improvement over existing solutions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

The problem investigated in this thesis is attribute-based access control with hidden policies and hidden credentials. An attribute-based access control policy makes a permit or deny decision based on the attributes of the entity requesting access, which are not limited to that entity's identity. For example, attributes of Alice may include the facts that she is a full-time student at the University of Pittsburgh, that her GPA is exactly 3.9, and that her GPA is greater than 3.5. In this thesis all attributes are assumed to be logical propositions; the access control policy is thus a Boolean formula consisting of attributes of the entity requesting access joined by logical connectives (AND, OR, and NOT).

Each attribute in an attribute-based access control policy must be associated with an entity trusted to make authoritative statements concerning its value. For example, the University of Pittsburgh's registrar's office is the authoritative source of information concerning the GPAs of university students. If a policy requires a GPA of at least 3.5, then it is not sufficient for Alice herself to claim that her GPA is 4.0; she must provide evidence from the registrar's office, the authority for the GPA attribute, that this is in fact the case. Such evidence may come in the form of a statement digitally signed with the registrar's private key, an example of a *digital credential*.

If an attribute-based access control policy is public and the access requester is willing to submit digital credentials proving that the policy is satisfied, then widely-deployed certificate technology suffices for policy enforcement [12]. Both the requester's attributes and the enforcer's policy can be considered private information, however, that should be protected as far as possible. Chapter 2 introduces the problem of *privacy-preserving* attribute-based access control by discussing the effect of various "communication models" on the confidentiality of attributes and policies. Briefly, if the access control policy is private but the credentials required of the access requester are accessible to the policy enforcer, then the enforcer can determine whether a requester satisfies a policy without disclosing it by retrieving the requester's credentials from a known repository or by querying the attribute authorities directly. When sensitive attributes of the requester such as income or ethnic origin should be kept hidden even from the policy enforcer, however, a message can be sent to the requester encrypted in such a way that it can be decrypted only by an entity with the required attributes. This message is either the requested resource or a secret key that enables an entity that satisfies the policy to gain access to the requested resource. Although this kind of cryptographic credential technology has not been widely deployed, a number of proposals can be found in the research literature [18, 23, 8]. Nevertheless, decrypting the message implies knowledge of at least one way of satisfying the access control policy, so the policy cannot truly be private.

A more sophisticated approach is for the access requester and policy enforcer to engage in a cryptographic protocol that takes a message and a policy from the enforcer as input and provides the message as output to the requester only if the requester's attributes satisfy the enforcer's policy, *without revealing the policy*. In fact, Frikken, Atallah, and Li proposed a "hidden policies with hidden credentials" protocol [14, 15] that reveals only an upper bound on the number of attributes in the policy, an upper bound on the number of credentials submitted by the access requester, and an upper bound on the size of the Boolean circuit representing the policy. To the best of our knowledge, this protocol represents the state of the art in attribute-based access control with hidden policies and hidden credentials. Chapter 3 discusses the techniques of Frikken, Atallah, and Li in detail and provides empirical evidence on the computational cost of the cryptographic operations involved, which proves to be substantial.

The rest of this thesis examines alternatives to Frikken, Atallah, and Li's solution that make use of online attribute providers or certification authorities (CAs). These are entities that can directly answer queries about a subject's attributes in a way similar to the "identity providers" in single sign-on solutions such as OpenID[1] and SAML[2] that manage the identities

[1]OpenID Web site: `http://openid.net/`
[2]SAML Web site: `http://www.oasis-open.org/committees/security/`

and attributes of users on behalf of other relying parties. The fundamental problem with the Frikken, Atallah, and Li protocol is that the access requester does not know which attributes are included in the policy, so every credential must be tried in combination with every policy attribute. By allowing the policy enforcer to directly query online CAs that can authoritatively provide attribute data, this major source of inefficiency is removed. The difficulty, however, is that the attributes of an access requester must still be kept private. This thesis presents protocols that reveal no information about the requester's attributes to the policy enforcer, while revealing varying amounts of structural information on the policy to the requester and CAs. More precisely, the policy enforcer remains completely oblivious to the outcome of the access control decision, while the access requester obtains a short message (such as a symmetric encryption key usable for decryption of additional data) if the policy is satisfied.

Chapter 4 investigates whether well-known secure multiparty computation techniques using Shamir secret sharing and arithmetic circuits in the private channel model are appropriate for attribute-based access control with online CAs. Our goal is for the policy enforcer and the CAs to jointly evaluate a function that produces an encryption of a message for the access requester's public key if and only if the policy is satisfied. Incorporating public key encryption into the arithmetic circuit itself is not practical, however, due to the size of the resulting circuit. We therefore show how an encrypted circuit output value can be obtained from Shamir shares of the arithmetic circuit's output by obliviously performing polynomial interpolation *outside of the circuit* using an additively homomorphic cryptosystem. Unfortunately, every CA needs to trust a majority of the protocol participants (the policy enforcer and the CAs), which constrains the range of policies that can be enforced. Extensive communication between the CAs is also necessary, which can lead to slow response times when the protocol is run over a wide-area network. We note that both of these problems can be mitigated if evaluation of the arithmetic circuit is delegated to a set of parties trusted to have an honest majority by all protocol participants.

Chapter 5 explores an approach to policy enforcement that uses homomorphic encryption to evaluate conjunctions and scrambled circuits to evaluate disjunctions. This eliminates the most computationally expensive parts of the Frikken, Atallah, and Li protocol (hidden credentials, private set intersection, and oblivious transfer). The conjunction evaluation protocol requires little computation, but involves as many rounds of communication as the number of conjuncts, so network latency can be a bottleneck when the protocol is run over a wide-area network. A latency/bandwidth trade-off is therefore described that only partially evaluates conjunctions homomorphically, moving the rest of the policy evaluation into a larger scrambled circuit. A simpler variant of the protocol of this chapter requires only a constant number of communication rounds and no scrambled circuit evaluation, but discloses both an upper bound on the number of conjunctions in the policy and an upper bound on the number of conjunctions that are satisfied.

1.1 SUMMARY OF CONTRIBUTIONS

The main contributions of this thesis are the following:

- We provide the first published empirical evaluation of the state-of-the-art protocol of Frikken, Atallah, and Li (FAL) for attribute-based access control with hidden policies and hidden credentials.
- We show how general secure multiparty computation techniques using arithmetic circuits can be applied to oblivious access control by moving public key encryption outside of the circuit using an additively homomorphic cryptosystem.
- We observe that hidden credentials, private set intersection, oblivious transfer, and scrambled circuit equality tests can all be eliminated from the FAL protocol by means of encrypted scrambled circuit inputs obtained from online CAs.
- We describe how evaluating conjunctions outside of the scrambled circuit using homomorphic encryption can further reduce the circuit size.

Together, these contributions substantially advance the state of the art in attribute-based access control with hidden policies and hidden credentials.

2.0 ATTRIBUTE-BASED ACCESS CONTROL PARADIGMS

This chapter provides context for the rest of the thesis by classifying attribute-based access control systems into four paradigms along the dimensions of proof of policy satisfaction versus oblivious access control and client-supplied credentials versus online attribute authorities. Figure 1 provides an overview of this scheme, which is discussed in detail in Section 2.1. Section 2.2 provides examples of concrete instantiations of the four paradigms, while Section 2.3 compares them in terms of credential and policy privacy.

2.1 THE FOUR PARADIGMS

In conventional access control systems, the policy enforcer obtains proof of policy satisfaction before granting access to a protected resource. The most obvious approach to attribute-based access control is thus to have the access requester provide any necessary credentials to the

	Client-supplied credentials	Online attribute authorities
Proof of policy satisfaction	Requester provides proof of attributes	Enforcer gathers proof of attributes
Oblivious access control	Oblivious policy enforcement with client credentials	Oblivious policy enforcement with online CAs

Figure 1: Attribute-based access control paradigms.

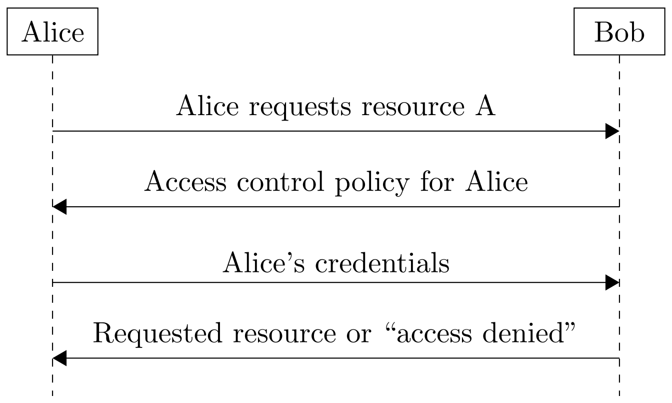
Alice
Bob
Alice requests resource A
Access control policy for Alice
Alice's credentials
Requested resource or "access denied"

Figure 2: Alice provides her digital credentials to Bob.

policy enforcer as proof of policy satisfaction, just as users commonly provide a password today as proof of identity. This is illustrated in Figure 2, which shows an access requester Alice submitting her digital credentials to a policy enforcer Bob, who determines whether Alice satisfies his policy on the basis of the submitted credentials. Bob will generally need to reveal at least the attributes in his policy to Alice so that she knows what credentials she must submit.

It is also possible for the policy enforcer to obtain proof of policy satisfaction from a source other than the access requester. For example, the policy enforcer can directly query authorities trusted to know the requester's attributes ("online CAs"), receiving the attribute values in reply. Of course, the attribute authorities must be willing to reveal these attribute values to the policy enforcer. This paradigm is shown in Figure 3, in which Bob queries CA 1 and CA 2 to determine whether predicates p and q are true of Alice, and uses the truth values to determine whether to return a requested message M to Alice or an indication that access is denied. Note that Alice does not need to provide any credentials. In fact, the ability of the policy enforcer to automatically retrieve any necessary credentials from online CAs or other credential repositories may be considered a convenience feature.

A very different access control paradigm can be seen in protocols in which the policy enforcer does not obtain proof of policy satisfaction at all (remaining *oblivious* to the outcome

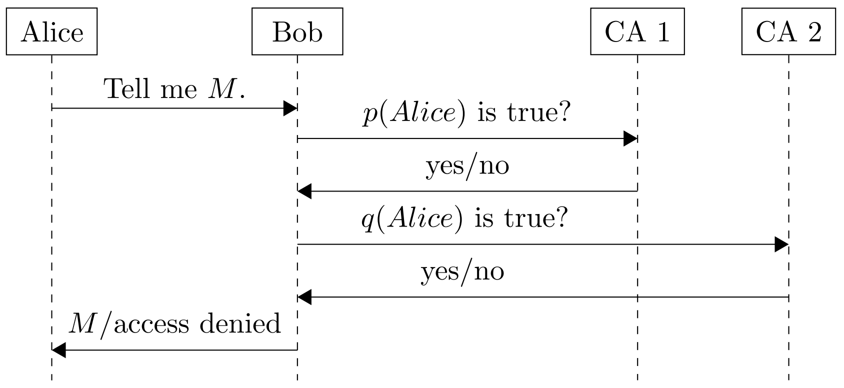

Figure 3: Bob learns Alice's credentials by querying online CAs.

of the access control decision). In fact, the policy enforcer does not need to learn anything about an access requester's attributes. The policy enforcer simply encrypts the requested resource in such a way that only requesters who satisfy the access control policy can decrypt it using their credentials. Since there is no subsequent communication from the requester to the enforcer, the enforcer learns nothing about the requester's credentials or whether the policy was satisfied. This situation is illustrated in Figure 4. Alice requests a message M from Bob, and receives a ciphertext that she can decrypt if she has the right cryptographic credentials.

A variation on the preceding paradigm is for the policy enforcer to obtain the access requester's cryptographic credentials from online CAs, encrypted for the requester's public

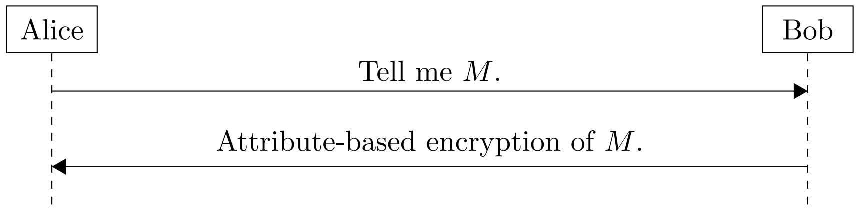

Figure 4: Alice uses cryptographic credentials to decrypt Bob's response.

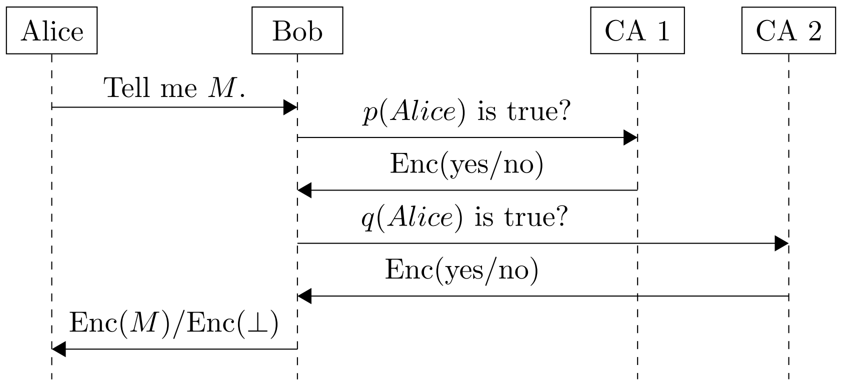

Figure 5: Alice receives an encrypted message only if she satisfies Bob's policy.

key, and then pass the encrypted credentials to the requester along with an encrypted response that implicitly encodes the access control policy as in the previous paradigm. This frees the requester from the need to have the right credentials available locally. A further development of this idea would be for the policy enforcer to use the requester's cryptographic credentials directly, while remaining oblivious to the outcome of the access control decision. Figure 5 depicts one way in which this might be accomplished. The policy enforcer Bob obtains encrypted responses from online CAs that encode Alice's attribute values without revealing them to him. Bob somehow processes these encrypted attribute values to obtain either the encryption of the requested message M for Alice's public key or the encryption of an unrelated value $\perp$, depending on whether the access control policy is satisfied. Bob is unable to distinguish between these two cases, however, because the final output for Alice is encrypted under her public key, just like her attribute values.

2.2 CONCRETE EXAMPLES

A few examples will show how technologies deployed in practice or proposed in the research literature fit into the classification scheme introduced in the previous section:

- An *X.509 public key certificate* [9] is a digitally signed document that binds a public encryption or signature verification key to a real-world identity (the *subject* of the certificate). An *X.509 attribute certificate* [12] includes verified attributes of the subject besides those that define the subject's identity. X.509 certificates are issued by trusted certification authorities (CAs) and can either be provided by the subject when requesting access to a resource, as in Figure 2, or obtained independently by a policy enforcer, as in Figure 3.

- An *oblivious attribute certificate* (OACert) [22] is like an X.509 attribute certificate except that it contains cryptographic *commitments* to attribute values instead of the subject's actual attribute values. These commitments can be used in zero-knowledge proof (ZKP) protocols [5, 11] that convince a verifier that an attribute value satisfies a condition without revealing any other information about the attribute value (for example, a subject might prove that her age is greater than 18 without revealing her exact age). Besides being a more powerful version of attribute certificates, OACerts can be used in an *oblivious commitment-based envelope* (OCBE) protocol [22, 21]. In OCBE, an access requester obtains a message only if a committed attribute value satisfies a specified predicate, but no information about the attribute value is revealed to the policy enforcer. This corresponds to the paradigm shown in Figure 4, except that the policy enforcer must have OACerts containing commitments to the attribute values used in the access control policy.

- In *multiauthority attribute-based encryption* (ABE) [7, 8], a ciphertext is associated with a set of attributes such that any user who has been issued decryption keys (possibly by different authorities) that correspond to a satisfying set of attributes can decrypt the message. No knowledge of the identities of potential recipients is needed at the time of message encryption. A policy enforcer can thus return a ciphertext encrypted according to the access control policy in reply to a request for the corresponding plaintext message, as in Figure 4, or the ciphertext can simply be made public.

- Another instantiation of the paradigm of Figure 4 is *hidden credentials* [18, 6]. In this system, a message is encrypted in such a way that specified digital credentials are needed to decrypt it. The identity of the intended recipient needs to be known at the time of encryption when this information is included in the recipient's digital credentials along

	Client-supplied credentials	Online attribute authorities
Proof of policy satisfaction	X.509 certificates (from the client), ZKP	X.509 certificates (from a repository)
Oblivious access control	OCBE, ABE, hidden credentials, Frikken, Atallah, and Li	**This thesis**

Figure 6: Concrete instantiations of attribute-based access control.

with relevant attribute values, as is usually the case. Since a hidden credential is a decryption key, it must be kept secret. The protocol of Frikken, Atallah, and Li described in Chapter 3 can be thought of as an extension of hidden credentials that requires additional rounds of interaction to provide better protection for the secrecy of the access control policy.

Figure 6 places the techniques discussed above within the scheme of Figure 1. The case of oblivious access control using online attribute authorities is the focus of this thesis, although it is not represented in the existing access control literature. The reasons for choosing such an approach will be presented in the following section.

2.3 COMPARISON OF PARADIGMS

In this thesis we assume that an access requester learns whether a requested message is obtained or not, and so it is impossible to prevent the requester from making inferences about the access control policy based on the outcome of the access control decision. All other information about the requester's credentials and the enforcer's policy can conceivably be concealed, however. For example, if the access requester is required to submit credentials to the policy enforcer without being told anything about the enforcer's policy, then no information is revealed about the access control policy except what can be learned from the enforcer's response: the requested resource or an indication that access is denied. The common practice of revealing the policy to access requesters is motivated by the inefficiency of having the requester submit every possibly relevant credential, along with the fact that requesters may be unwilling to submit credentials unless they know that they satisfy the policy.

These practical problems are addressed in the paradigm of Figure 3 (the policy enforcer obtains credentials from online CAs), which still reveals only the minimum level of policy information to the access requester, although the CAs now learn which attributes are part of the policy. From the viewpoint of an access requester, this model is likely to be more practical than having the policy enforcer demand credentials without providing an accompanying policy, except for the fact that policy enforcers must be granted free access to the credentials of possible requesters. Of course, a hybrid model in which credentials come from both the access requester and other online sources is also possible.

Perfect privacy for the requester's attributes clearly requires oblivious access control, since the fact that a policy is satisfied reveals information about the attributes of any entity that satisfies it. Ideally, oblivious access control reveals no information at all about the requester's attributes, and this is possible in the paradigm of Figure 4 (oblivious access control with client cryptographic credentials), but this approach reveals more information about the access control policy than logically necessary. If a client supplies credentials to use in making an access request, then the client immediately knows a satisfying set of credentials for the policy if access is granted.

This problem can be solved with the paradigm of Figure 5 (oblivious access control with encrypted replies from online CAs). The access requester now sees nothing except the result of the access control process (the requested message or "access denied"), while the policy enforcer only obtains encrypted attribute values that reveal no information (in a computational sense). This paradigm is the approach taken by the new solutions described in this thesis. We are not aware of prior work in oblivious access control using encrypted replies from online CAs.

3.0 THE SOLUTION OF FRIKKEN, ATALLAH, AND LI

In Frikken, Atallah, and Li's protocol (FAL for short) [15], a requester Alice wishes to obtain a secret message from Bob, who has an access control policy that is a function of Alice's Boolean attributes. Bob should not learn anything about Alice's attributes, not even whether his policy is satisfied, while Alice should learn as little as possible about Bob's policy. FAL builds upon hidden credentials [18], so Alice is assumed to possess a hidden credential for each of her attributes.

Recall that a major limitation of hidden credentials is that while Alice's credentials are never revealed to Bob, Alice necessarily learns one way of satisfying Bob's policy if she can decrypt his message. In essence, Alice has keys and if she can decrypt Bob's response, she knows which keys she used. This is the problem that Frikken, Atallah, and Li set out to solve. FAL gives Alice a new set of keys for each access request. She does not know which credentials her keys correspond to, and must use all of the keys together in order to have any chance of decrypting Bob's response, which she can only do if her attributes satisfy Bob's policy.

Sections 3.1–3.4 of this chapter describe each of the steps of FAL in detail, while Section 3.5 provides the first published performance evaluation of an implementation of the protocol. We not only demonstrate that the protocol is quite expensive computationally, but also provide insight into the relative cost of each stage and how these costs scale with the number of attributes in Bob's policy and the number of credentials used by Alice. In discussing FAL, we assume that the CHP 3 variant of Section 3.1.3 in Frikken, Atallah, and Li's paper is used, as it comes closest to the ideal of completely hiding Alice's credentials and Bob's policy.

3.1 PROTOCOL OVERVIEW

The high-level organization of the FAL protocol is illustrated in Figure 7. Alice first engages in a *credential hiding protocol* with Bob that gives her a secret value r_i if and only if she has attribute i and some other random value otherwise, where the r_i values are chosen by Bob. Bob does not learn whether Alice obtains r_i. Alice then uses the outputs of the credential hiding protocol as inputs to a *scrambled circuit* generated by Bob that gives her the requested message M if Bob's policy is satisfied (that is, if Alice provides the bits of r_i as the values of the circuit input wires that correspond to attribute i for a set of attributes that satisfies Bob's policy). The credential hiding phase of the protocol can be further divided into two parts: hidden credentials decryption, which produces a value β_{ij} for each combination of the ith attribute in Bob's policy and the jth credential held by Alice, and private set intersection, which reduces the β_{ij} values for each attribute i to a single value η_i that is supplied as input to the scrambled circuit. Section 3.2 describes the hidden credentials stage of FAL in greater detail, Section 3.3 describes the private set intersection stage, and Section 3.4 describes the scrambled circuit evaluation stage.

3.2 HIDDEN CREDENTIALS

A hidden credential is a private decryption key that corresponds to a public encryption key bound to both an identity and an attribute. In conventional public key encryption, a CA certifies that a public key belongs to a certain user, and that messages encrypted for that public key can only be decrypted by that user. In a hidden credentials system, a public key belongs to a particular user having a certain attribute, and messages encrypted for the public key can only be decrypted by the user if he or she has the specified attribute. For example, a message can be encrypted such that Alice can decrypt it only if she has a GPA of at least 3.5. This is guaranteed by the fact that the public encryption key is derived from the identity of the credential issuer (CA), along with Alice's identity and the attribute of having a GPA of at least 3.5. The CA can use its private key to compute the corresponding private

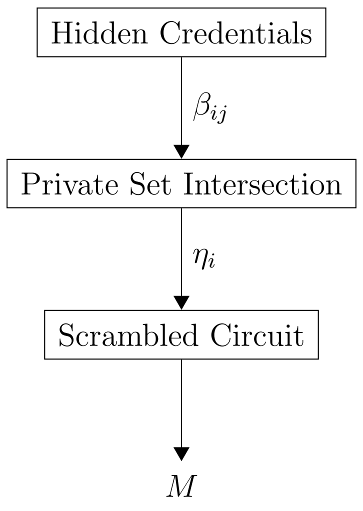

Figure 7: Overview of Frikken, Atallah, and Li's protocol.

decryption key, which it will only give to Alice if she does have a GPA of at least 3.5. If Alice keeps her hidden credentials secret, then other parties, knowing nothing about Alice's attributes, can encrypt a message for her that she can decrypt only if she has a specified attribute.

In our experimental evaluation of the FAL protocol, we implement hidden credentials using a variant of the Boneh-Franklin identity-based encryption (IBE) scheme [4] proposed by Ivan and Dodis [19]. This is a slight simplification of the BasicIdent scheme of [4]. It is based on a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ that satisfies the Decisional Bilinear Diffie-Hellman Assumption [19], where G_1 and G_2 are groups of large prime order. The bilinearity of the map means that $e(u^a, v^b) = e(u, v)^{ab}$ for all $u, v \in G_1$ and all $a, b \in \mathbb{Z}$. Let g be a generator of G_1, let q be the order of G_1, and let $h : \{0, 1\}^* \rightarrow G_1$ be a hash function that maps arbitrary strings to elements of G_1. The scheme is defined as follows:

CA Setup: Randomly pick a secret key $s \in \mathbb{Z}_q$, and publish g^s as the CA's public key.

Extract$(ID, attr, s)$**:** The decryption key that corresponds to identity ID and attribute $attr$ is extracted using the CA's master secret s as $h(ID \,||\, attr)^s$.

Encrypt$(M, g^s, ID, attr)$**:** The encryption of message $M \in G_2$ for identity ID and attribute

$attr$ controlled by a CA with public key g^s is $(U, V) = (g^r, M \cdot e(h(ID \,||\, attr)^r, g^s))$, for a random $r \in \mathbb{Z}_q$.

Decrypt$(U, V, h(ID \,||\, attr)^s)$**:** The ciphertext (U, V) is decrypted using secret decryption key $h(ID \,||\, attr)^s$ as $V/e(h(ID \,||\, attr)^s, U) = M$.

The above encryption scheme is IND-ID-CPA secure [4]. Informally, this means that a polynomial-time adversary cannot distinguish (with probability significantly better than 1/2) between the encryptions of two messages of its choice for an $(ID, attr)$ pair of its choice if it does not possess the decryption key $h(ID \,||\, attr)^s$. In a hidden credentials system, a CA will only issue the credential $h(ID \,||\, attr)^s$ to a user with identity ID possessing attribute $attr$, so a message encrypted for this $(ID, attr)$ pair can only be decrypted if the user with identity ID does have attribute $attr$.

When a monotone access control policy consisting of Boolean attributes joined by AND and OR operators is not sensitive, hidden credentials alone can be used to enforce access control over a message M. Suppose that the policy is written in disjunctive normal form. Then each conjunction (disjunct) is associated with a copy of M that is split between the conjuncts in such a way that M can be recovered when all of the conjuncts are true, but not otherwise. Specifically, for a conjunction of n conjuncts, n values are randomly chosen whose product is M, and each value is encrypted for a different attribute in the conjunction. A requester who has all of the attributes can decrypt the n ciphertexts, compute the product of the plaintext values, and obtain M.

In the FAL protocol, Alice has a hidden credential from the appropriate issuer for each of her m attributes. For each of the n attributes in his policy, Bob chooses a random key $k_i[0]$ and encrypts $k_i[0]$ for the public key corresponding to the hidden credential that Alice would have if she possessed attribute i, producing ciphertext α_i. These ciphertexts are sent to Alice. Alice attempts to decrypt each ciphertext α_i ($i \in [1, n]$) using each of her hidden credentials in turn, obtaining a plaintext value β_{ij} that is equal to $k_i[0]$ if her jth credential proves that she has attribute i (and is issued by the right CA), and is a random value otherwise. Alice does not learn if any of her $m \cdot n$ β_{ij} values is equal to any of Bob's $k_i[0]$ values.

3.3 PRIVATE SET INTERSECTION

For each attribute in his policy, Bob chooses a random key $k_i[1]$ and engages in a private set intersection protocol with Alice. This protocol takes $k_i[0]$ and $k_i[1]$ as inputs from Bob and the β_{ij} values as input from Alice for $j \in [1, m]$. It produces $k_i[1]$ as Alice's output η_i if $k_i[0]$ matches one of the β_{ij} values supplied by Alice, and produces a random output value for Alice otherwise. In other words, Alice's m β_{ij} values are reduced to a single value η_i that may be the key $k_i[1]$ proving that she has attribute i. The set intersection protocol used in FAL was described in [13] and is summarized below.

The basic idea is that Alice constructs a polynomial

$$(x - \beta_{i1})(x - \beta_{i2}) \cdots (x - \beta_{im}) = \sum_{j=0}^{m} a_j x^j.$$

Bob evaluates this polynomial at the point $x = k_i[0]$ and obtains 0 if any one of Alice's β_{ij} values is equal to $k_i[0]$ and some nonzero value otherwise. Bob multiplies the result that he obtains by a freshly chosen random value so that it is randomized if nonzero. He then adds $k_i[1]$ and returns the sum to Alice.

The obvious problem with this scheme is that Bob learns whether Alice obtains $k_i[1]$ (that is, whether she has attribute i). In order to prevent this, Bob actually evaluates the polynomial obliviously using a homomorphic cryptosystem so that he does not know the result of evaluating Alice's polynomial at $x = k_i[0]$, multiplying it by a random value, and adding $k_i[1]$. In our implementation, we use the Paillier cryptosystem [28]:

Key Gen: A key pair is generated by choosing $n = pq$, where p and q are large primes, and $g \in \mathbb{Z}^*_{n^2}$ such that $\gcd(L(g^\lambda \bmod n^2), n) = 1$, where $\lambda = \text{lcm}(p-1, q-1)$ and $L(u) = (u-1)/n$. The public key is (n, g) and the private key is (λ, μ), where $\mu = L(g^\lambda \bmod n^2)^{-1} \bmod n$.

Encrypt(m, n, g)**:** The encryption of message $m \in \mathbb{Z}_n$ for public key (n, g) is $c = g^m \cdot r^n \bmod n^2$ for a random $r \in \mathbb{Z}_n$.

Decrypt(c, λ, μ)**:** The ciphertext c is decrypted using private key (λ, μ) as $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$.

The security of the Paillier cryptosystem is based on the Decisional Composite Residuosity Assumption [28], which posits that it is computationally intractable to distinguish numbers of the form $y^n \bmod n^2$ for $y \in \mathbb{Z}^*_{n^2}$ from other members of $\mathbb{Z}^*_{n^2}$. This cryptosystem has several useful homomorphic properties:

- For any ciphertext $c = \mathit{Encrypt}(m, n, g)$, $c^k \bmod n^2$ is an encryption of $k \cdot m \bmod n$.
- For any ciphertext $c = \mathit{Encrypt}(m, n, g)$, $c \cdot g^k \bmod n^2$ is an encryption of $m + k \bmod n$.
- For any ciphertexts $c_1 = \mathit{Encrypt}(m_1, n, g)$ and $c_2 = \mathit{Encrypt}(m_2, n, g)$, $c_1 \cdot c_2 \bmod n^2$ is an encryption of $m_1 + m_2 \bmod n$.

Taken together, these three properties imply that given ciphertexts $c_1, \ldots, c_n$ produced by encrypting plaintexts $m_1, \ldots, m_n$ for the same public key, a new ciphertext can be computed that is the encryption of any linear combination $k_1 \cdot m_1 + \cdots + k_n \cdot m_n$ of the plaintext values. This computation does not require knowledge of the private decryption key.

In particular, recall that Alice has the coefficients of a polynomial $p(x)$ that Bob is supposed to evaluate at a value known only to him. Alice encrypts each coefficient for a Paillier public key generated by her (for which only she knows the private key). The value $p(k_i[0])$ of Alice's polynomial at the point $k_i[0]$ can be computed by Bob as a linear combination of the encrypted coefficients supplied by Alice. He can then randomize the encrypted result if it is nonzero (Alice does not have attribute i) by homomorphically multiplying it by a random constant $r \in \mathbb{Z}_n$, and finally add $k_i[1]$ homomorphically to the encrypted value. The resulting ciphertext is returned to Alice, who decrypts it to obtain $k_i[1]$ if she has attribute i. A minor detail is that Bob can ensure that the polynomial $\sum_{j=0}^{m} a_j x^j$ that he evaluates has nonzero coefficients by assuming that $a_m = 1$.

3.4 SCRAMBLED CIRCUIT EVALUATION

At this point, for each attribute i in Bob's policy, Alice has a value η_i that is $k_i[1]$ if she has attribute i or a random value otherwise (note that neither she nor Bob can tell which of these possibilities is the case). Bob now generates a scrambled circuit [24] that performs a

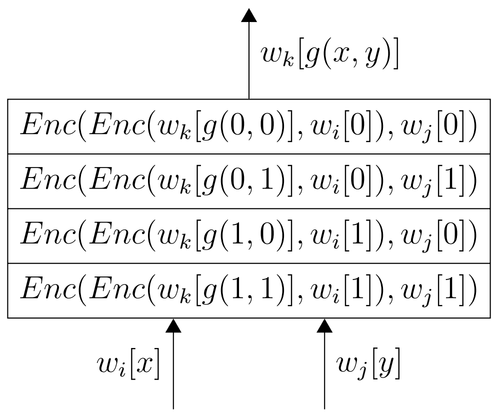
$w_k[g(x,y)]$
$Enc(Enc(w_k[g(0,0)], w_i[0]), w_j[0])$
$Enc(Enc(w_k[g(0,1)], w_i[0]), w_j[1])$
$Enc(Enc(w_k[g(1,0)], w_i[1]), w_j[0])$
$Enc(Enc(w_k[g(1,1)], w_i[1]), w_j[1])$
$w_i[x]$
$w_j[y]$

Figure 8: Scrambled circuit binary gate.

bit-by-bit comparison between Alice's η_i values and his $k_i[1]$ values. Such a circuit consists of wires and gates. Each wire is associated with two random keys: one that corresponds to Boolean true and one that corresponds to Boolean false. The circuit evaluator (Alice) knows exactly one key for each wire, and does not know the mapping between the keys and the Boolean value of the wire, which is chosen, along with the keys, by the circuit generator (Bob).

A binary gate consists of four ciphertexts: one encrypted using each combination of the keys associated with the two input wires. Upon reaching a binary gate, the circuit evaluator is able to decrypt exactly one of the ciphertexts using the keys that she knows for the gate's input wires. Decryption produces one of the keys associated with the gate's output wire. This is either the key that corresponds to true or the key that corresponds to false, depending on the truth table of the gate. The circuit evaluator cannot tell which one was obtained, however, due to the fact that the keys are selected at random by the circuit generator.

For example, suppose that a gate takes wires i and j as input and generates the value of wire k as output. Let the keys associated with the false and true values for wire i be $w_i[0]$ and $w_i[1]$, respectively, and similarly for wires j and k. If Alice has $w_i[0]$ for the value of wire i and $w_j[1]$ for the value of wire j, she decrypts the appropriate entry in the gate's "encrypted truth table" using $w_i[0]$ and $w_j[1]$ as decryption keys. Depending on the type of gate (e.g., AND or OR), she will obtain either $w_k[0]$ or $w_k[1]$ (see Figure 8). For instance, if the gate is an AND gate, she obtains $w_k[0]$, and if the gate is an OR gate, she obtains $w_k[1]$. In order for Alice to know which encrypted truth table entry to decrypt, Bob can associate the wire values $w_i[0]$ and $w_i[1]$ with public marker values 0 or 1 in arbitrary order (and similarly for the other wires). If Alice has a value marked 0 for wire i and a value marked 0 for wire j, she decrypts the first entry in the encrypted truth table. If she has values marked 0 and 1, she decrypts the second entry, and so on. The decrypted value for wire k will also be marked 0 or 1, but this does not have any relationship to the true Boolean value of the wire.

Thus Alice can evaluate each interior gate in the scrambled circuit for which she already knows the input wire values. The values of the circuit input wires are obtained by running a 1-out-of-2 oblivious transfer protocol [27] with Bob. Since Bob generates the circuit, he knows the wire values for both true and false. Alice is allowed to learn only one of these, although she can pick which one. Furthermore, Bob is not allowed to learn which value Alice picks. This is precisely the functionality called 1-out-of-2 oblivious transfer.

Using oblivious transfer for the circuit input wires, Alice can evaluate a circuit generated by Bob on a private input supplied by her and obtain the values of the circuit's output wires. The circuit structure is known to Alice, but each gate looks the same regardless of the underlying truth table. As stated earlier, Alice's inputs to the circuit in the FAL protocol are the bits of an η_i value for each attribute i in Bob's policy. Bob constructs the circuit to perform a bit-by-bit comparison between Alice's inputs and his $k_i[1]$ values. The result of each comparison is a wire whose truth value corresponds to that of the proposition "Alice has attribute i." These wires are then used as inputs to the AND and OR gates representing Bob's policy. The circuit has a single output wire. If this wire is logically true, then the wire value obtained by Alice is the message that she requested and on which Bob wants to enforce his access control policy. If the circuit output wire is false, its value is not related to the requested message and informs Alice that access is denied.

In general, a scrambled circuit does not impose any restrictions on the number of gates or how they are connected, except that each gate may be evaluated only once (the circuit must be acyclic). Circuit input wires and intermediate gate outputs can serve as an input to multiple gates. It is convenient, however, to imagine a circuit representing an access control policy written in disjunctive normal form as a binary tree with OR gates at the top and AND gates at the bottom. Attributes that appear in more than one conjunction can either be represented by a single circuit input wire, or duplicate copies of the attribute can be used in the prior stages of the FAL protocol, each one supplying the value of a different circuit input wire. The latter option obviously adds additional overhead, while the former reveals information about the policy structure unless a wire representing every attribute appears in every conjunction (the truth tables of the gates evaluating a conjunction can be programmed to ignore the values of irrelevant input wires).

The FAL protocol as described above reveals the exact number of attributes in Bob's policy and the exact number of credentials used by Alice. It is easy for Bob to pretend to be interested in attributes that are not part of his policy, however, and Alice can pretend to have more credentials than she actually has. Bob can also pad the scrambled circuit with unnecessary gates so that it reveals only an upper bound on the size of his policy.

3.5 IMPLEMENTATION AND PERFORMANCE EVALUATION

The FAL protocol as described in Sections 3.2–3.4 was benchmarked to test its practicality. The hidden credentials system was implemented in C using the PBC pairing-based cryptography library, which in turn relies on the GNU Multiple Precision Arithmetic Library (GMP).[1] The "type A" pairing provided by the PBC library was used with the default parameters. This is based on an elliptic curve group with an order of 160 bits, where the curve is defined over a field whose order is 512 bits, providing approximately 80 bits of security (comparable to a 1024-bit RSA key [1]). The private set intersection protocol used an implementation of the Paillier cryptosystem based on GMP with a 1024-bit modulus (n). The polynomial to be evaluated was generated using the modular polynomial arithmetic facilities of the NTL library.[2] The Fairplay [26] system was used for scrambled circuit evaluation with $k_i[1]$ values

[1]PBC Web site: `http://crypto.stanford.edu/pbc/`; GMP Web site: `http://www.gmplib.org/`
[2]NTL Web site: `http://www.shoup.net/ntl/`

of 80 bits and the fastest oblivious transfer protocol supported (type 4, based on [27] with communication batching). The key sizes mentioned above favor speed over security [20]. Bob's policy was conservatively set to be a conjunction of all of the attributes in his policy (a more elaborate policy would require a larger scrambled circuit).

Experiments were conducted using two machines with 2.33 GHz Intel Xeon E5410 processors on a LAN. The machines ran the 32-bit Ubuntu 10.04 server distribution and had library versions GMP 4.3.2, PBC 0.5.8, and NTL 5.5.2 installed. No attempt was made to parallelize any of the operations to take advantage of the multicore processor.

Table 1 shows the average duration of each stage of the protocol over 10 trials as well as the total response time for 8, 16, and 32 client credentials and policies consisting of 8, 16, and 32 attributes. The "Other" category includes all network communication outside of Fairplay. As can be seen, most of the time is consumed by private set intersection and scrambled circuit evaluation. The cost of generating and evaluating the encrypted polynomials for private set intersection is approximately proportional to the product of the number of policy attributes and the number of credentials, while the cost of scrambled circuit evaluation is proportional to the number of policy attributes.

Even though much of the cryptography was implemented using the highly optimized assembly language routines of the GMP library, it is apparent that the FAL protocol is computationally demanding, with a total time in excess of 14 seconds even for 8 attributes and 8 credentials. Although the response time of the protocol can be improved by performing cryptographic operations in parallel on a multicore system and by pipelining the protocol stages (for example, Alice can begin hidden credential decryption while Bob is still engaged in hidden credential encryption), this does not change the fact that *seconds* of server CPU time must be dedicated to each client.

It should also be kept in mind that in order to effectively hide the actual number of attributes and credentials used by Alice and Bob, the number of attributes and credentials in the protocol execution must be higher than the real values. The propositional nature of the attributes in a hidden credentials system also has a tendency to increase the number of attributes or credentials required. Suppose that Bob wishes to test whether Alice's GPA is greater than 3.0, and includes this as a single attribute in his policy. Then Alice, not knowing

Table 1: Frikken, Atallah, and Li protocol performance (seconds).

	8 policy attributes		
	8 creds.	16 creds.	32 creds.
Hidden credential encryption	0.181	0.178	0.180
Hidden credential decryption	0.399	0.760	1.494
Encrypted polynomial generation	1.717	3.333	6.711
Encrypted polynomial evaluation	0.920	1.722	3.334
Polynomial result decryption	0.100	0.100	0.104
Fairplay	10.885	10.872	10.871
Other	0.026	0.024	0.023
Total	14.228	16.989	22.717
	16 policy attributes		
	8 creds.	16 creds.	32 creds.
Hidden credential encryption	0.333	0.334	0.332
Hidden credential decryption	0.758	1.497	2.946
Encrypted polynomial generation	3.417	6.734	13.379
Encrypted polynomial evaluation	1.832	3.447	6.671
Polynomial result decryption	0.202	0.203	0.202
Fairplay	21.546	21.614	21.604
Other	0.026	0.033	0.036
Total	28.114	33.862	45.170
	32 policy attributes		
	8 creds.	16 creds.	32 creds.
Hidden credential encryption	0.648	0.642	0.641
Hidden credential decryption	1.484	2.953	5.868
Encrypted polynomial generation	6.733	13.421	26.903
Encrypted polynomial evaluation	3.664	6.893	13.341
Polynomial result decryption	0.402	0.405	0.401
Fairplay	43.055	43.030	43.110
Other	0.045	0.056	0.091
Total	56.031	67.400	90.355

the GPA threshold (or even that her GPA is required) might need to supply credentials for $\text{GPA} > 2.5, \text{GPA} > 2.6, \ldots, \text{GPA} > 3.9$. Alternatively, Bob could include policy attributes for $\text{GPA} \in (3.0, 3.1], \text{GPA} \in (3.1, 3.2], \ldots, \text{GPA} \in (3.9, 4.0]$, and combine these using a disjunction. In either case, the number of attributes or credentials involved is substantially increased.

4.0 POLICY EVALUATION USING SHAMIR SHARES

One of the major sources of inefficiency in the protocol of Frikken, Atallah, and Li is that Alice must supply every credential available to her if she wants to ensure that she gains access whenever possible, while the hidden credential decryption and private set intersection parts of the protocol have computational cost roughly proportional to the number of credentials supplied (and not the number of credentials actually needed). It is therefore natural to ask whether it is possible to construct a more efficient access control solution that allows Bob to obtain precisely the attribute values needed for policy enforcement from the CAs responsible for those attributes, without actually learning Alice's attribute values.

In this chapter, we consider an approach to oblivious policy enforcement in which the policy enforcer engages in a distributed computation with online CAs that produces a message encrypted for the access requester's public key if and only if the policy is satisfied, without revealing to the enforcer whether the ciphertext that is the protocol's output is actually the requested message. For policy evaluation, we will use the techniques for evaluating an arithmetic circuit on Shamir shares originally developed in [2], following the clearer presentation of [17]. This background material is covered in Section 4.1.

Generating a circuit output encrypted for the access requester's public key without performing public key encryption within the circuit is a nontrivial problem, however. Section 4.2 presents our solution: oblivious polynomial interpolation of encrypted shares using an additively homomorphic cryptosystem. This technique is the main contribution of this chapter, since it reduces the cost of generating an encrypted circuit output to little more than the cost of evaluating the circuit itself. Section 4.3 evaluates the performance of this construction, and Section 4.4 describes its limitations and compares it with FAL. Figure 9 displays our high-level approach.

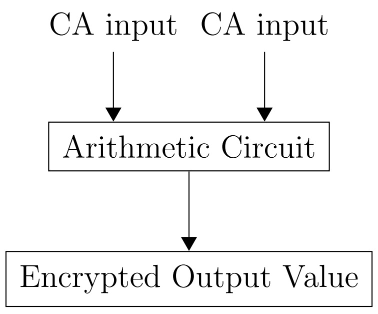

Figure 9: Policy enforcement with an arithmetic circuit.

4.1 COMPUTING WITH SHAMIR SHARES

Shamir's threshold secret sharing scheme [29] allows a value to be split into n shares such that the original value can be recovered from any t of those shares. It is based on polynomials over the finite field $\mathbb{Z}_p$, where p is a prime number larger than n. In order to share the secret value $s \in \mathbb{Z}_p$, a random degree $t-1$ polynomial $p(x)$ is chosen such that $p(0) = s$ (that is, the constant term is s while the other coefficients are selected uniformly from $\mathbb{Z}_p$). The ith share is simply $p(i)$. Knowledge of at least t shares allows $p(0)$ to be computed by interpolation, while knowledge of fewer than t shares leaves $p(0)$ completely undetermined. In fact, as a special case of Lagrange interpolation,

$$p(0) = \sum_{i=1}^{t} \left((-1)^{i+1} \cdot \binom{t}{i} \cdot p(i) \right).$$

The secure multiparty computation protocol of [2] requires private and authenticated communication channels between each pair of participants; these can be approximated in practice using SSL connections and public key infrastructure. Each private input to the computation is shared with all n participants using Shamir secret sharing with a threshold of $t < n/2 + 1$ (this means that t dishonest parties can obtain all of the private inputs, breaking the security of the protocol). Computation on the private inputs is represented by an "arithmetic circuit" in which wires have values from $\mathbb{Z}_p$ and gates perform addition and multiplication on their input wire values. On encountering a gate, every party holds a share of the value of each gate input wire. A protocol allows each party to obtain a share of the value of the gate's output wire, which can serve as input to another gate. When shares of the circuit's output wires have been computed, the actual circuit outputs can be revealed by interpolation of t shares. None of the intermediate wire values can be obtained without the cooperation of at least t participants, however, since all wire values are shared among all participants with a threshold of t.

For evaluating a Boolean access control policy, the secret wire values will be 0 for false and 1 for true. An AND gate computes the conjunction of two input wire values x and y as $x \cdot y$, while a NOT gate computes the negation of an input wire value x as $1 - x$. Each circuit input wire corresponds to an attribute, and the CA responsible for that attribute will share its value—0 or 1—with all of the protocol participants (that is, the CAs and the policy enforcer). How the Boolean result of evaluating the policy—a wire whose value is 0 or 1—can be converted into the encryption of a message for the access requester's public key will be discussed later. First, we describe the mechanics of evaluating a gate.

In a NOT gate with input wire value x, party i holds $p(i)$, where $p(0) = x$. In order to obtain a share of the output wire value $1 - x$, party i simply computes $1 - p(i)$. This works because if $p(y)$ is a degree $t - 1$ polynomial and $p(0) = x$, then $q(y) = 1 - p(y)$ is also a degree $t - 1$ polynomial and $q(0) = 1 - p(0) = 1 - x$. Furthermore, party i can locally compute the ith share of $1 - x$, which is just $q(i) = 1 - p(i)$.

Multiplication gates are more complicated, and require interaction between the parties. Suppose that $a(z)$ and $b(z)$ are two degree $t-1$ polynomials such that $a(0) = x$ and $b(0) = y$, and party i holds $a(i)$ and $b(i)$. Party i now computes $c_i = a(i) \cdot b(i)$ and shares c_i with all of the other parties using threshold t. Let c_{ij} be the share of c_i given to party j, that is, $q_i(j)$ for some degree $t - 1$ polynomial $q_i(z)$ such that $q_i(0) = c_i$. Party j then locally computes

$$d_j = \sum_{i=1}^{n} \left((-1)^{i+1} \cdot \binom{n}{i} \cdot c_{ij} \right).$$

It happens to be a fact that d_j is a share of the output wire value $x \cdot y$ with threshold t, that is, $d_j = q(j)$ for some degree $t - 1$ polynomial $q(z)$ such that $q(0) = a(0) \cdot b(0) = x \cdot y$ [17].

4.2 GENERATING AN ENCRYPTED OUTPUT VALUE

The protocols for evaluating AND gates and NOT gates given above are sufficient to enable the evaluation of any Boolean formula from Shamir-shared attribute values, since $x \vee y$ is equivalent to $\neg(\neg x \wedge \neg y)$. Obtaining a Boolean access control decision alone is insufficient, however. If it is revealed to the policy enforcer, then information about the requester's attribute values is disclosed. In theory, it is possible for the policy enforcer to supply the requested message and the requester's public key as inputs to the circuit, and for the circuit to perform public key encryption on either the message or an unrelated value (such as 0), depending on the outcome of policy evaluation, but public key encryption requires a very large circuit. (If this were not the case, public key operations would not be expensive on an ordinary processor).

Our solution to this problem is to multiply the Boolean access control decision by the requested message m within the arithmetic circuit (the policy enforcer supplies m as a circuit input), which produces the wire value $r = m$ if the policy is satisfied and $r = 0$ otherwise. Party i therefore obtains a share r_i of r. Recall that $r = \sum_{i=1}^{t} \gamma_i \cdot r_i$, where $\gamma_i = (-1)^{i+1} \cdot \binom{t}{i}$. If party i for $i \in [1, t]$ encrypts $\gamma_i \cdot r_i$ for a Paillier public key generated by the message requester and sends the ciphertext to the policy enforcer, then the product of these ciphertexts can be computed by the enforcer, yielding an encryption of r by the homomorphic property of the Paillier cryptosystem (see Section 3.3). This ciphertext is forwarded to the message requester for decryption. In other words, t shares of r are encrypted and the polynomial interpolation is done obliviously by the policy enforcer using homomorphic cryptographic operations. (This does not rely on any peculiarity of Paillier. Another semantically secure encryption scheme with an additive homomorphism and a sufficiently large message space would also work.)

The security of the arithmetic circuit evaluation protocol will not be proven here, as it is a well-known construction [2]. The fact that the policy enforcer cannot derive any meaningful information about the plaintexts of the Paillier-encrypted shares from the ciphertexts follows directly from the semantic security of the Paillier encryption scheme [28]. (Semantic security means that a polynomial-time adversary has negligible advantage in computing any function of the plaintext corresponding to a given ciphertext, compared to a simulator that is not given the ciphertext at all. Thus replacing all ciphertexts by encryptions of random values that truly do not reveal any information would not significantly affect the behavior of the adversary.) The message requester receives a single Paillier ciphertext and upon decrypting it obtains either the requested message or 0. This reveals no information about how the ciphertext was generated, let alone the policy represented by the arithmetic circuit, except for whatever can be inferred from the result of the access control decision itself.

It is not necessary to assume that the message requester's Paillier public key is known in advance to all of the CAs. A new one can be generated on demand by the requester and signed using any signature scheme that is verifiable by the CAs. The signed public key is then distributed to the CAs by the policy enforcer. Generating a fresh Paillier key pair in this fashion has the advantage of making the encrypted shares obtained by the enforcer useless outside of the current interaction. (If there were a non-negligible probability that a plaintext encrypted for one public key could be decrypted to a related value using any one of a polynomially bounded number of randomly generated public/private key pairs, then a polynomial-time adversary would be able to break the semantic security of the encryption scheme merely by generating new key pairs at random.)

4.3 IMPLEMENTATION AND PERFORMANCE EVALUATION

When arithmetic circuit evaluation is complete, the CAs can encrypt their Shamir shares of the circuit output wire in parallel, and then t ciphertexts need to be multiplied together by the policy enforcer. On the machines used in all of the experiments described in this thesis, with 2.33 GHz Intel Xeon E5410 processors running in 32-bit mode, one Paillier encryption takes about 25 ms using the GMP 4.3.2 library for big integer arithmetic and a 1024-bit Paillier modulus. One modular multiplication takes less than a millisecond. Since t is at most one greater than half of the number of CAs providing attribute values, computing an encrypted output value from shares of the arithmetic circuit's output wire is a small, fixed cost. The rest of this section therefore focuses on the cost of evaluating the circuit itself.

Arithmetic circuit evaluation using Shamir shares was tested in VIFF, the Virtual Ideal Functionality Framework.[1] This is a secure multiparty computation system written in Python (the snapshot of May 31, 2010 was used). A notable feature of VIFF is that operations are automatically parallelized: arithmetic circuit gates are evaluated whenever their operands are available. The passive runtime module was selected; this uses algorithms that are secure against semihonest adversaries when a majority of the parties are fully honest (semihonest adversaries follow the protocol but may seek to extract private information). VIFF also offers a runtime module that is secure against malicious adversaries (that may not follow the protocol) when more than 2/3 of the parties are honest, but the requirement that each CA must trust 2/3 of the other CAs seems undesirable. It should also be noted that when using the passive runtime module, a smaller number of malicious adversaries than the Shamir secret sharing threshold cannot extract a CA's private input; the requirement of semihonest behavior is needed only to ensure that the correct output value is computed. This means that the policy enforcer must trust every CA to be semihonest, but a CA does not have to trust every other CA to be semihonest. For additional discussion of the security properties of VIFF, the reader is referred to [16].

Arithmetic was performed over the finite field $\mathbb{Z}_p$ for an 80-bit prime p (this is assumed to be sufficiently large for a message containing a symmetric key). Experiments were conducted using 5, 9, and 17 parties to the computation, one being the policy enforcer and the rest being CAs. The Shamir secret sharing threshold was set to $(n+1)/2$, where n is the number of parties.

Because evaluating a multiplication gate in the arithmetic circuit requires each party to send shares to every other party, substantial network communication is involved. The experiments below were therefore conducted in both local-area network (LAN) and wide-area network (WAN) configurations. Each party ran in a virtual machine instance obtained from the Amazon Elastic Compute Cloud (Amazon EC2) service.[2] The instances were of the "High-CPU Medium Instance" type, provisioned with two cores of a 2.33 GHz Intel Xeon E5410 CPU and 1.7 GB of memory, and ran the 32-bit Ubuntu 10.04 server distribution.

[1]VIFF Web site: `http://viff.dk/` (see also [16])
[2]Amazon EC2 Web site: `http://aws.amazon.com/ec2/`

Amazon EC2 instances can be placed in one of four regions: US East (Northern Virginia), US West (Northern California), EU (Ireland), and Asia Pacific (Singapore). In the LAN experiments, all parties were placed in the US East region and typically experienced a round-trip time to other parties of less than one millisecond. In the WAN experiments, the parties were geographically distributed as follows:

- When there are 5 parties, 2 are located in the US East region, and 1 is located in each of the other three EC2 regions.
- When there are 9 parties, 3 are located in the US East region, and 2 are located in each of the other three EC2 regions.
- When there are 17 parties, 5 are located in the US East region, and 4 are located in each of the other three EC2 regions.

Average round-trip times of up to 280 ms were observed between the Asia Pacific region and the US East and EU regions.

This WAN configuration seems to be representative of any geographic distribution with a similar maximum round-trip time between a pair of players. For example, Figure 10 compares the LAN and WAN configurations for 17 parties described above with a configuration in which 16 parties are located in the US East region but 1 party is located in the Asia Pacific region. The arithmetic circuit being evaluated is a complete binary tree of AND gates. Each CA (every party except for the policy enforcer) supplies one input to the circuit, which may be the input to multiple gates. (This is not meaningful when all of the gates are AND or OR gates, but is realistic when the policy consists of a disjunction of conjunctions that share some attributes.) Response time is measured from the point of view of the policy enforcer and includes SSL connection setup. The error bars show 90% confidence intervals for the mean. As can be seen from the nearly indistinguishable top two lines of the plot, performance in the "one party over WAN" setting in which every party but one is located on a LAN is no better than the case in which all parties have to communicate with most of the other parties over a WAN. Because every party needs to communicate with every other party in evaluating a multiplication gate, the slowest party determines the response time for everyone.

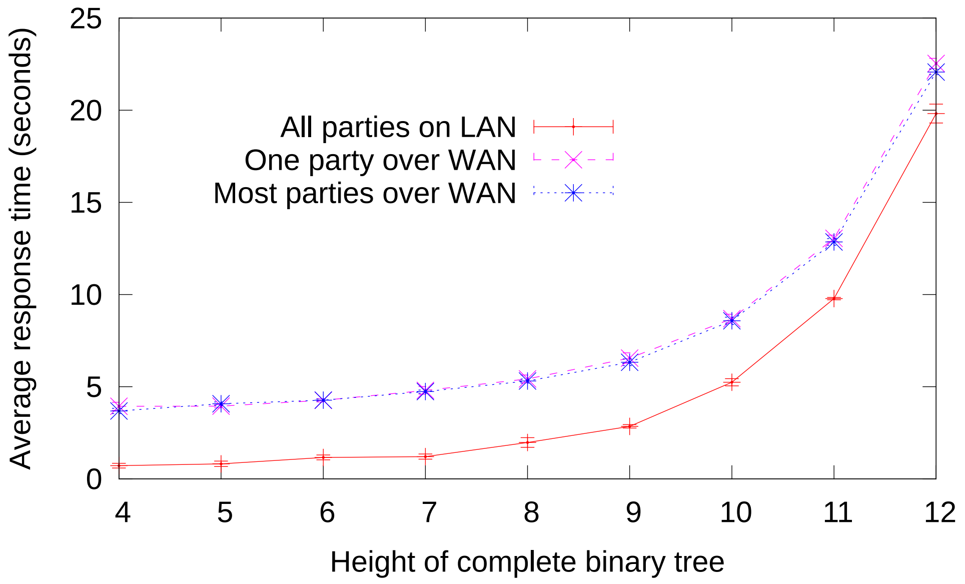

Figure 10: VIFF LAN/WAN performance comparison. Response times are shown for 17 parties located in Northern Virginia, 16 parties in Northern Virginia and 1 party in Singapore, and 4–5 parties in each of the four EC2 regions (average of 10 runs).

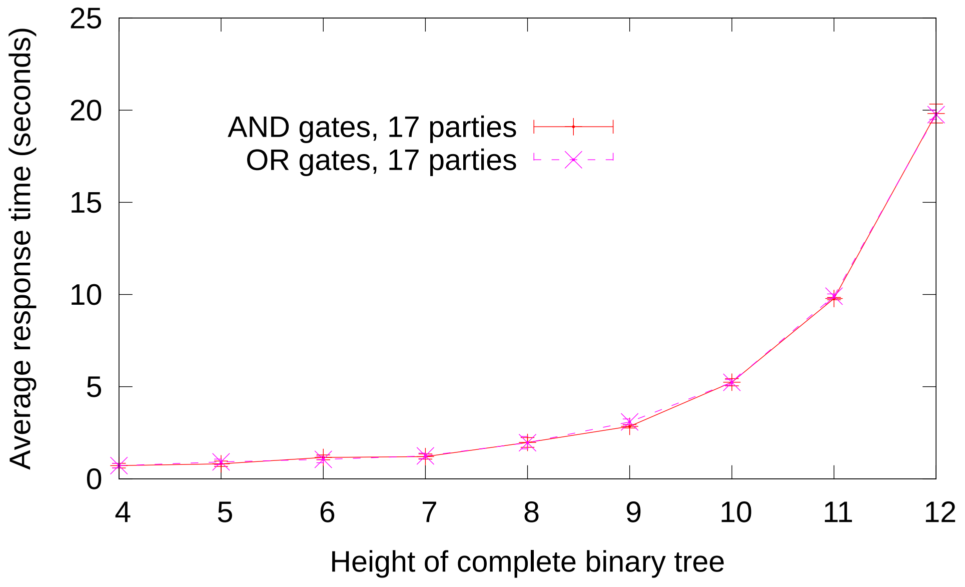

Figure 11: VIFF AND/OR gate performance comparison on a LAN (average of 10 runs).

Figures 11 and 12 compare the performance of evaluating a complete binary tree of AND gates with evaluating a complete binary tree of OR gates over a LAN and over a WAN, respectively. There is very little difference between AND and OR gates, since they both involve one multiplication, while an OR gate adds several negations, which can be computed through local arithmetic operations on shares and require no network communication (see Section 4.1). Given the almost negligible cost of negations, the rest of this section will present results for binary trees of AND gates.

Figures 13 and 14 describe the performance of evaluating binary trees of AND gates over a LAN for a varying number of protocol participants. Each CA supplies one input to the circuit, so for a small number of CAs, it is not meaningful to consider large circuits in policy evaluation. Figure 14 shows the average time spent per gate, revealing that there is a large setup cost that can be amortized over the size of the circuit. For a given circuit size, a larger number of protocol participants brings greater complexity (but the secret sharing threshold is higher, possibly yielding better security). Figures 15 and 16 show analogous data for the WAN case, which is more realistic unless every party is on the same LAN.

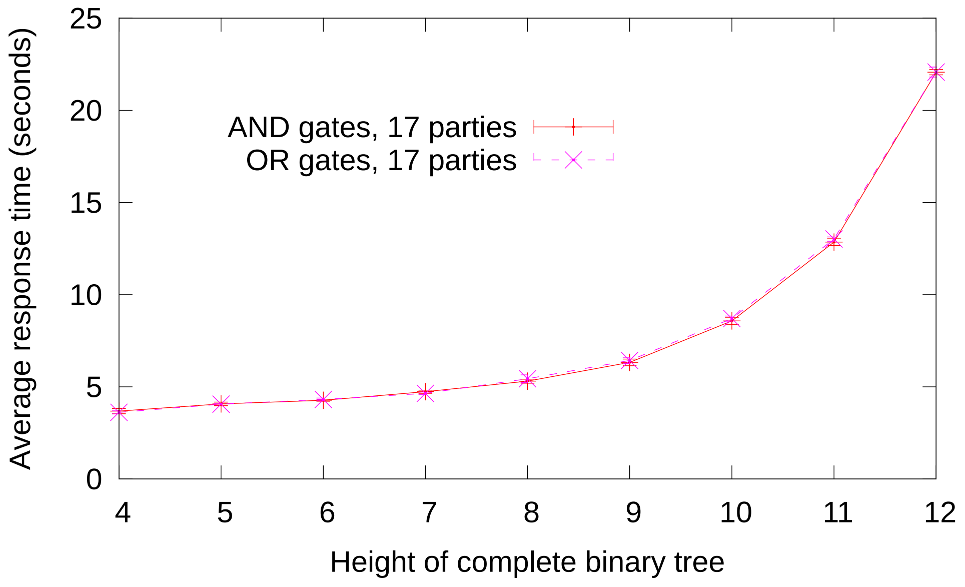

Figure 12: VIFF AND/OR gate performance comparison on a WAN (average of 10 runs).

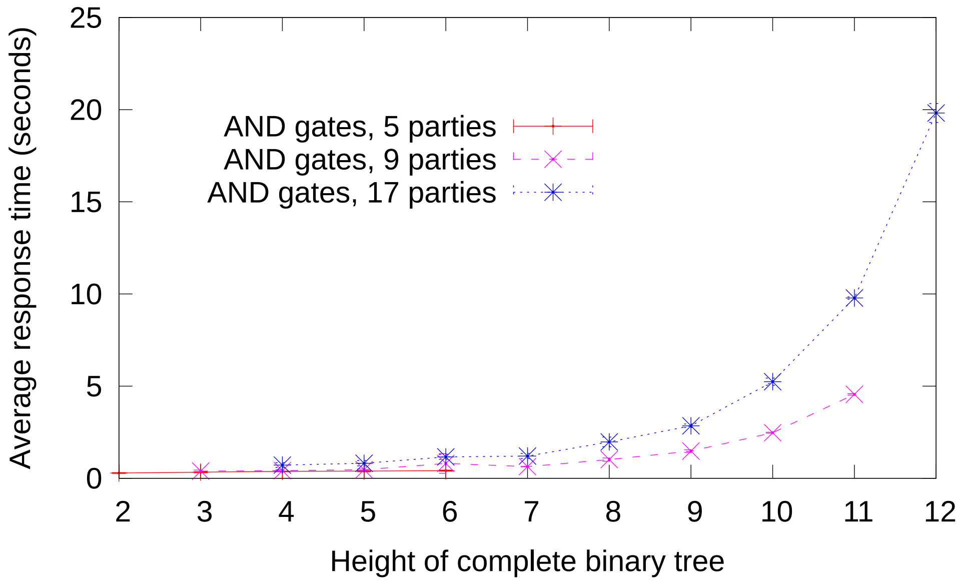

Figure 13: VIFF response times on a LAN (average of 10 runs).

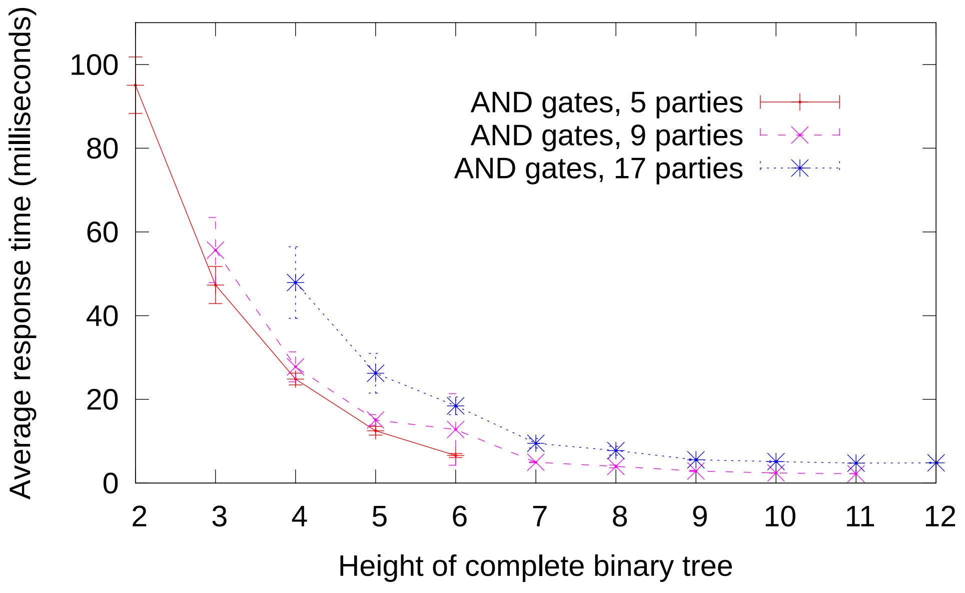

Figure 14: VIFF response time per gate on a LAN (average of 10 runs).

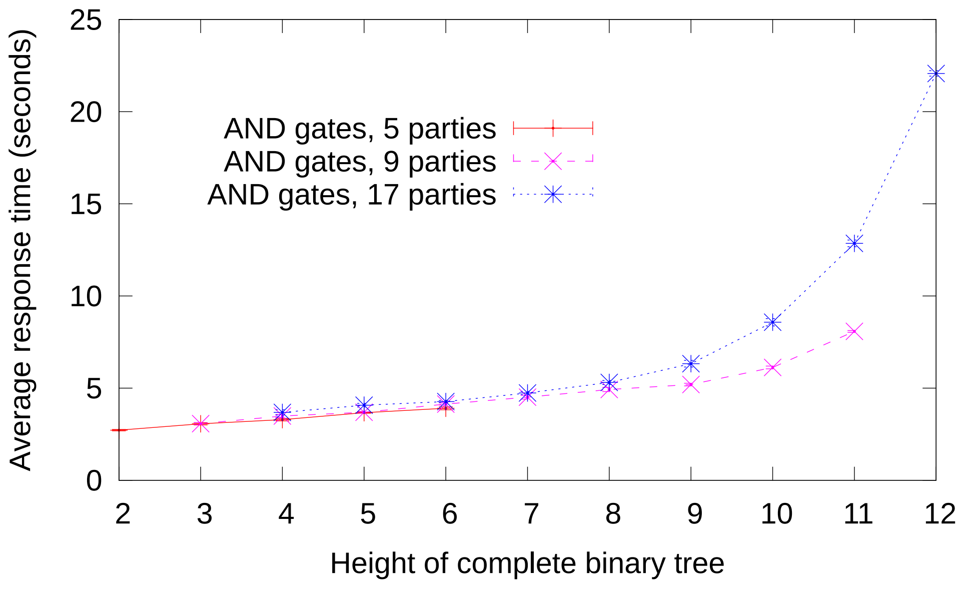

Figure 15: VIFF response times on a WAN (average of 10 runs).

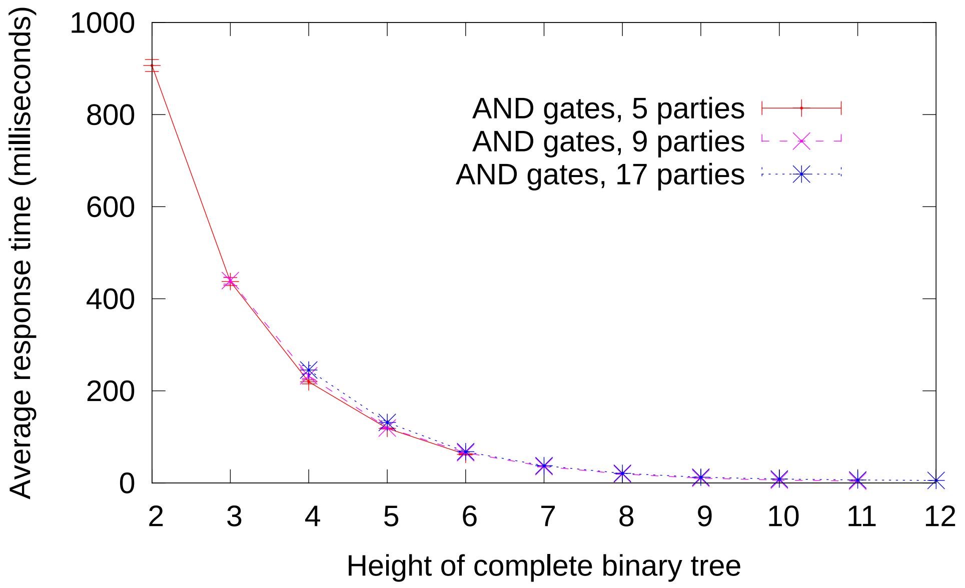

Figure 16: VIFF response time per gate on a WAN (average of 10 runs).

4.4 DISCUSSION

Evaluating an arithmetic circuit on Shamir-shared input values requires the circuit to be known to all of the parties, because the action taken for each gate depends on the specific type of gate (AND, NOT, etc.). This clearly limits the privacy of an access control policy represented by the circuit. At the cost of a constant-factor expansion in the circuit size, however, the function of each gate can be concealed by adding additional circuit inputs supplied by the policy enforcer that "program" the gates. For example, suppose that a policy consists of AND and OR gates arranged in a binary tree. The fact that the circuit has a binary tree structure can be made public, but the policy enforcer wishes to hide which gates are AND gates and which gates are OR gates. For any binary gate with inputs x and y, this can be done by adding a "control line" z that selects between $x \wedge y$ and $x \vee y$ as the output of the gate, which is the result of evaluating the expression

$$((x \wedge y) \wedge z) \vee ((x \vee y) \wedge \neg z).$$

Since the actual value of z during circuit evaluation is Shamir-shared with all of the parties by the policy enforcer, only a cooperating subset of the CAs as large as the secret sharing threshold t can extract the type of each gate. Note also that the arithmetic circuit itself does not reveal which specific attributes the circuit inputs correspond to, but only the identities of the CAs providing the input values.

It is necessary for the CAs to know the identities of the other CAs, since each CA needs to trust more than half of the n parties involved in evaluating the circuit (the Shamir secret sharing threshold t needs to be below $n/2 + 1$ in order for the protocol for multiplication gates to function correctly). This requirement of mutual trust between CAs is a serious drawback that goes far beyond the assumption that the policy enforcer trusts every CA. Even when there is insufficient trust between CAs, however, the CAs and the policy enforcer may be able to agree on a set of parties that are mostly trusted by all participants, and these trusted parties can carry out the circuit evaluation on shares of input values provided by the original protocol participants.

For instance, suppose that the policy enforcer and CAs are able to agree on 5 parties that are all known to be semihonest, and of which at least 3 are believed to be fully honest by each participant. Then the CAs can share their private circuit input values with the 5 trusted parties using a secret sharing threshold of $t = 3$, and the policy enforcer can similarly share the values of any additional inputs that control the functionality of the circuit gates. The trusted parties evaluate each gate in the circuit using the protocols described earlier in Section 4.1, and finally send encrypted shares of the circuit output wire value to the policy enforcer for interpolation using an additively homomorphic cryptosystem, as described in Section 4.2. Because the trusted parties only manipulate Shamir-shared values, a coalition of 3 dishonest parties is required to learn either the private inputs or the private output of the computation. If all of the trusted parties are located on a local-area network, there are also obvious performance advantages, as can be seen in Figure 10.

Figure 10 also reveals that as the circuit size increases, computation dominates communication in determining the total protocol response time (the gap between the LAN and WAN lines narrows). Since VIFF is implemented in Python, it is reasonable to expect that a substantial speedup can be achieved by careful implementation of the performance-critical parts of the code in C (or even compilation of Python to native machine code). For smaller policies, the response time can be reduced noticeably by maintaining long-term network connections between the protocol participants, which is practical if a policy enforcer repeatedly engages in the protocol with the same group of CAs. For these reasons, the performance data reported in this chapter should not be regarded as the major obstacle to the practicality of the protocol for policies of moderate size; the requirement of mutual trust between protocol participants is more likely to limit its applicability.

In any case, even the current performance of VIFF represents a dramatic computational improvement over the FAL results reported in Section 3.5. On a LAN, a conjunction of 8 attributes requires an average of 0.390 seconds to evaluate in VIFF, while the same conjunction requires at least 14 seconds in FAL (assuming that at least 8 client credentials are submitted). A conjunction of 16 attributes requires an average of 0.719 seconds to evaluate in VIFF, versus at least 33 seconds in FAL (assuming that at least 16 client credentials are submitted).

5.0 HOMOMORPHIC CONJUNCTION EVALUATION

The main weaknesses of the secure multiparty computation protocol of the previous chapter are that every CA needs to trust a majority of the protocol participants and that the structure of the policy being enforced (if not the specific attributes utilized) is revealed to every CA. Ideally, a CA should not need to know anything about the other CAs involved in policy enforcement, and should not learn anything about the policy except what attribute values it supplies. This is possible if a CA is queried directly by the policy enforcer, returns a reply encrypted for the public key of the access requester, and the encrypted reply is used obliviously in policy enforcement. This chapter takes this approach, evaluating conjunctions using a homomorphic cryptosystem and disjunctions using a scrambled circuit.

Sections 5.1–5.2 present the homomorphic cryptosystem and the basic conjunction evaluation protocol. In Section 5.3, we observe that hidden credentials, private set intersection, oblivious transfer, and scrambled circuit equality tests can all be eliminated from the FAL protocol by means of encrypted scrambled circuit inputs obtained from online CAs. We then describe how evaluating conjunctions outside of the scrambled circuit using homomorphic encryption further reduces the circuit size. Section 5.4 discusses a latency/bandwidth trade-off involved in applying homomorphic conjunction evaluation. Section 5.5 presents performance evaluation results, while Section 5.6 fills in a number of practical details.

The access control policy will be assumed to be in disjunctive normal form (DNF). A negated attribute can be rewritten as an attribute that expresses the logical negation of the original attribute ("Alice is single" instead of "Alice is not married"), so negations of policy attributes will not be considered. Private and authenticated communication channels are assumed to exist between the policy enforcer and the CAs. The CAs are also assumed to know (or to be able to verify) the message requester's public key.

5.1 HOMOMORPHIC TAG-BASED ENCRYPTION

The encryption scheme used in this chapter is a tag-based variant of the Cramer-Shoup cryptosystem [10] proposed by MacKenzie, Reiter, and Yang [25]. This scheme is defined over a group G_q of large prime order q in which the Decisional Diffie-Hellman Assumption [3] is believed to hold. One such group may be found by generating a large prime p and then choosing a large prime q that divides $p-1$. G_q can then be defined as the (unique) subgroup of order q in $\mathbb{Z}_p^*$, the multiplicative group of integers modulo p. Given such a group G_q with generator g, the scheme is defined as follows (where the notation $\in_R$ denotes the choice of a random element from some group):

Key Gen: Choose $g_2 \in_R G_q$ and $a, b, c, d, e \in_R \mathbb{Z}_q$, and set $U = g^a(g_2)^b$, $V = g^c(g_2)^d$, and $W = g^e$. The public key is defined as $\langle g, g_2, U, V, W\rangle$ and the private key is defined as $\langle a, b, c, d, e\rangle$.

Encrypt$(M, \langle g, g_2, U, V, W\rangle, t)$**:** Choose $r \in_R \mathbb{Z}_q$ and let $x = g^r$, $y = (g_2)^r$, $w = W^r M$, and $v = U^r V^{rt}$. Return $\langle x, y, w, v\rangle$ as the ciphertext.

Decrypt$(\langle x, y, w, v\rangle, \langle a, b, c, d, e\rangle, t)$**:** If $v \neq x^{a+ct}y^{b+dt}$, fail. Otherwise, return w/x^e.

$\otimes$**:** $\langle x, y, w, v\rangle \otimes \langle x', y', w', v'\rangle = \langle x \cdot x', y \cdot y', w \cdot w', v \cdot v'\rangle$

The Decisional Diffie-Hellman Assumption states that it is computationally infeasible to distinguish a Diffie-Hellman triple (g^x, g^y, g^{xy}) from a non-Diffie-Hellman triple (g^x, g^y, g^z) for random $x, y, z \in \mathbb{Z}_q$.

The tag is the parameter t supplied to the encryption and decryption algorithms. It functions in many ways as part of the public key, since the same tag must be used in encryption and decryption. However, given known public key parameters, any agreed-upon integer in $\mathbb{Z}_q$ can be used as the tag. This means that an essentially unlimited number of public keys can be derived from one set of public key parameters. In this chapter, a session ID will be used as the tag for all cryptographic operations, which keeps ciphertexts generated in one session from being usable in another session. This property is captured formally as the TNM-CCA2 security of the cryptosystem [25], which means that a polynomial-time adversary given a ciphertext has negligible advantage in generating a ciphertext encrypted with a different tag whose plaintext value has any relation to that of the ciphertext given to the adversary, compared to a "simulator" that is not given the first ciphertext at all. TNM-CCA2 security implies semantic (IND-CPA) security, because if the adversary could extract any information about the plaintext value corresponding to a ciphertext without knowledge of the private decryption key, then it could generate the encryption of a related value under a different tag by the usual encryption algorithm.

Suppose that two ciphertexts $c = \langle x, y, w, v \rangle$ and $c' = \langle x', y', w', v' \rangle$ are the encryptions of messages M and M', respectively, for the same public key $\langle g, g_2, U, V, W \rangle$ and tag t. Then the homomorphic combination operation $c \otimes c'$ computes an encryption of $M \cdot M'$ without knowledge of the private key because the following four equations hold:

$$\begin{aligned} x \cdot x' &= g^r \cdot g^{r'} = g^{r+r'} && (5.1) \\ y \cdot y' &= (g_2)^r \cdot (g_2)^{r'} = (g_2)^{r+r'} && (5.2) \\ w \cdot w' &= W^r M \cdot W^{r'} M' = W^{r+r'} \cdot M \cdot M' && (5.3) \\ v \cdot v' &= U^r V^{rt} \cdot U^{r'} V^{r't} = U^{r+r'} V^{(r+r')t} && (5.4) \end{aligned}$$

5.2 CONJUNCTION EVALUATION USING HOMOMORPHIC COMBINATION

Let the plaintext value 1 stand for Boolean true, and any other plaintext value—selected uniformly at random from the message space—stand for Boolean false. Then a conjunction of encrypted inputs can be computed obliviously by reducing the input ciphertexts to a single output ciphertext through repeated application of the homomorphic combination operator $\otimes$. This produces an encryption of the product of all the plaintext values, which is 1 (true) if all of the plaintext values were 1, and an encryption of a random element of the message space otherwise, which is 1 only with negligible probability.

In the context of access control, suppose that the policy consists of a single conjunction of Boolean attributes. The policy enforcer can query online CAs for the truth values of the attributes in the conjunction. Each CA returns the encryption of 1 for the message requester's public key if the attribute is true, and the encryption of a random element of the message space otherwise (all encryptions use a common session ID as the tag). The policy enforcer computes the homomorphic combination of all of the encrypted attribute values, which is the encryption of 1 only when all of the attributes are true. This result can then be homomorphically combined with an encryption of the requested message M, producing the encryption of M when the conjunction evaluates to true, and the encryption of a random value otherwise. When this ciphertext is sent to the message requester and decrypted, it yields the desired result of the access control protocol—the requested message if and only if the policy is satisfied, and no meaningful information about the message otherwise, assuming that valid messages can be distinguished with high probability from random values (for example, by being padded with zeros).

The policy enforcer obtains only encrypted values from CAs, so no information about the actual attribute values is revealed. CAs interact only with the policy enforcer and thus do not need to know anything about the other CAs. Since conjunction evaluation is performed by local manipulation of encrypted values by the policy enforcer, the only information about the policy disclosed to other entities is that the contacted CAs learn the requested attributes and the message requester learns whether access was permitted. If the tag used for each conjunction evaluation is freshly chosen by the message requester and is never reused, then the ciphertexts obtained by the policy enforcer cannot be used outside of the current session for unintended purposes.

We published the preceding protocol in a short paper [30], along with a policy cycle resolution algorithm that will not be discussed here. This work only considered policies consisting of a single conjunction of attributes, however. One approach to supporting disjunctions is to write the access control policy in disjunctive normal form, and then run a separate instance of the protocol for each conjunction (disjunct). True conjunctions will produce an encryption of the requested message, while false conjunctions will produce the encryption of a random value. In order to hide the actual number of conjunctions in the DNF formula, the policy enforcer can add spurious conjunctions to the policy that are never true, as well as make duplicate copies of conjunctions that may be true. Nevertheless, this approach reveals an upper bound on the total number of conjunctions, and an upper bound on the number of conjunctions that are true. The next section shows how homomorphic conjunction evaluation can be combined with scrambled circuit evaluation to reveal only an upper bound on the total number of conjunctions.

5.3 DISJUNCTION EVALUATION USING A SCRAMBLED CIRCUIT

As explained in Section 3.4, scrambled circuit evaluation allows a message requester to compute the output of any Boolean circuit from private input values without learning the values of intermediate wires in the circuit. For each input wire, the circuit evaluator needs to obtain exactly one of the two possible keys that correspond to Boolean true and Boolean false. Most of the complexity of the FAL protocol consists in computing the correct input wire values for a circuit that evaluates the access control policy without revealing which policy attributes the input wires correspond to.

If the circuit generator (policy enforcer) is allowed to query online CAs, then the correct value for each input wire can be chosen by a CA responsible for the corresponding attribute and encrypted for the circuit evaluator's public key. This encrypted wire value is returned to the policy enforcer, who simply forwards it to the message requester along with an indication of which circuit input wire it is intended for (no oblivious transfer is necessary, because it is the CA, and not the requester, that supplies the circuit input value). The message requester can then decrypt the value of each circuit input wire and proceed to evaluate the circuit without ever learning the meaning of the wire in the policy. Furthermore, this approach can be seamlessly combined with the FAL protocol by supplying the message requester with some input wire values through the policy enforcer's interaction with online CAs and with others through the hidden credential and private set intersection techniques of FAL, providing backward compatibility with client-side cryptographic credentials.

The main problem with this scheme is that all policy evaluation takes place in the scrambled circuit. In order to preserve the privacy of the policy, a larger circuit should be generated than necessary, and it should follow some standard circuit structure, such as a complete binary tree, that reveals only an upper bound on the size of the policy. (Spurious inputs that are not used in policy evaluation can be generated directly by the policy enforcer.) But large circuits take more time to generate, transmit over a network, and evaluate.

Although the multiplicative homomorphic property of the cryptosystem described in Section 5.1 can be used only to evaluate conjunctions, and not disjunctions, it can be used in combination with scrambled circuit evaluation to enforce any monotone policy of AND and OR operators, as shown below. Performing the conjunction evaluation locally at the policy enforcer through homomorphic operations keeps the size of the scrambled circuit small, and does not even reveal an upper bound on the size of the conjunctions.

A straightforward application of the conjunction evaluation technique described in the preceding section is not sufficient, however. Representing Boolean false as an arbitrary element of the message space other than 1 means that the value obtained by the message requester for a false conjunction is unpredictable to the policy enforcer. On the contrary, scrambled circuit evaluation requires a true conjunction to reveal the circuit input wire value that corresponds to Boolean true, and a false conjunction to reveal the circuit input wire value that corresponds to Boolean false. Both of these values must be predictable because they are used in circuit generation, and furthermore, they should not be distinguishable from one another by the message requester so that the truth value of the conjunction is not disclosed.

The solution presented below is to augment homomorphic combination of encrypted values with a second protocol that does the reverse, in a sense: if the conjunction is true, then the encryption of a random element of the message space is produced; otherwise, an encryption of 1 is produced. When the policy enforcer runs these two protocols in parallel, they each yield a ciphertext encrypted for the message requester's public key. Exactly one of these will be the encryption of 1, and the other will be the encryption of a random value. These two ciphertexts are homomorphically combined with encryptions of the true scrambled circuit input wire value and of the false wire value, respectively. One of these is obscured by multiplication with a random value, while the other is left unchanged by multiplication with 1. The two resulting ciphertexts are randomly permuted and sent to the message requester. Whichever one can be decrypted to a properly formed input wire value (distinguishable through padding) is the value to be used for the circuit input wire.

```
1: $M \leftarrow w_i[1]$
2: $pk \leftarrow$ public key of message requester
3: $t \leftarrow$ encryption tag (session ID) chosen by message requester
4: $c \leftarrow Enc_{pk,t}(M)$
5: **for all** $attr \in conjunction$ **do**
6:     $ca \leftarrow$ online CA responsible for $attr$
7:     $c \leftarrow c \otimes QueryCA(ca, attr, pk, t)$
8: **return** $c$
```

Figure 17: Algorithm for disclosing wire value $w_i[1]$ if a conjunction of attributes is true. $QueryCA(ca, attr, pk, t)$ returns $Enc_{pk,t}(1)$ if $attr$ is true, and $Enc_{pk,t}(r)$ for a random message r otherwise.

In other words, the policy enforcer (circuit generator) has wire values $w_i[0]$ and $w_i[1]$, corresponding to Boolean false and Boolean true for wire i. The message requester should learn $w_i[0]$ if a conjunction of attribute values is false and $w_i[1]$ if the conjunction is true. The policy enforcer runs two protocols with the online CAs. The first produces a ciphertext c_1 that is the encryption of $w_i[1]$ if the conjunction is true and is the encryption of a random value otherwise (this protocol is shown in Figure 17 and was described in Section 5.2 above). The second protocol produces a ciphertext c_2 that is the encryption of $w_i[0]$ if the conjunction is false and is the encryption of a random value otherwise (this protocol is shown in Figure 18 and will be described in greater detail below). After running these protocols, the policy enforcer sends their output values c_1 and c_2 to the message requester in random order. The requester then decrypts c_1 and c_2, obtaining $w_i[1]$ along with a random plaintext value if the conjunction is true, and $w_i[0]$ along with a random plaintext value if the conjunction is false. It is assumed that $w_i[0]$ and $w_i[1]$ can be distinguished from random elements of the message space through appropriate padding, but not from each other, given only one of them, since they are random keys of the same length. The message requester uses whichever of $w_i[0]$ and $w_i[1]$ is obtained as the value of wire i in evaluating the scrambled circuit.

```
1: $M \leftarrow w_i[0]$
2: $pk \leftarrow$ public key of message requester
3: $t \leftarrow$ encryption tag (session ID) chosen by message requester
4: $r \leftarrow$ random element of the message space
5: $c \leftarrow Enc_{pk,t}(r)$
6: for all $attr \in conjunction$ do
7:     $ca \leftarrow$ online CA responsible for $attr$
8:     $c \leftarrow QueryCA(ca, attr, pk, t, c)$
9: return $c \otimes Enc_{pk,t}(M)$
```

Figure 18: Algorithm for disclosing wire value $w_i[0]$ if a conjunction of attributes is false. $QueryCA(ca, attr, pk, t, c)$ returns $Enc_{pk,t}(1)$ if $attr$ is false, and $c \otimes Enc_{pk,t}(1)$ otherwise.

The protocol that produces an encryption of $w_i[0]$ whenever the conjunction is false (Figure 18) works as follows. The policy enforcer initializes ciphertext c to be the encryption of a random element of the message space for the requester's public key. CAs whose attributes appear in the conjunction are queried sequentially by the policy enforcer and passed c as part of the query. If the CA's attribute is true, then it returns the homomorphic combination of c with a fresh encryption of 1. Otherwise, it returns a fresh encryption of 1. The policy enforcer sets c to be the value returned by the CA and queries the next CA with c until every CA in the conjunction has been queried. After every CA has been queried, c remains the encryption of the random value chosen by the policy enforcer if every attribute in the conjunction is true, and is replaced by an encryption of 1 otherwise. Finally, the policy enforcer homomorphically combines c with an encryption of $w_i[0]$.

Because the policy enforcer must trust the CAs to provide correct attribute values, it is reasonable to assume that the CAs are also trusted to follow the protocol as specified (that is, they are semihonest or passive adversaries). A CA that does not follow the protocol can be caught cheating during an audit in which the ciphertexts in its reply are decrypted, and its reputation would suffer even if no direct punitive action is taken. Note that the policy enforcer is unable to tell whether a CA's attribute is true by inspecting the returned ciphertext because homomorphic combination of a ciphertext with a fresh encryption of 1 leaves the plaintext value unchanged, but randomizes the ciphertext so that it becomes a fresh random encryption of the plaintext value (see equations (5.1) through (5.4)). By the semantic security of the cryptosystem, the policy enforcer cannot distinguish a fresh encryption of the random value that it initially chose from a fresh encryption of 1.

The TNM-CCA2 security of the encryption scheme [25] implies that besides revealing no information in a computational sense, the ciphertexts seen by the policy enforcer and CAs are useless outside of the current access control session if the encryption tag is never reused by the message requester. The requester obtains exactly one value to use for a scrambled circuit input wire (along with a uniformly distributed random element of the message space). Since wire values are symmetric encryption keys selected uniformly at random, this wire value does not reveal the logical Boolean value of the wire or anything about the attributes in the conjunction. An elaborate formal proof of the security of the rest of the scrambled circuit construction can be found in [24].

The privacy properties of the homomorphic conjunction evaluation protocols introduced in this chapter can be summarized in the following theorem:

Theorem 1. *The protocols of Figures 17 and 18 reveal no information to the CAs except the attributes they are queried about, the public key of the message requester, and a session ID (encryption tag). No new information about the message requester's attributes is revealed (in a computational sense) to the policy enforcer and CAs.*

Proof. Each CA learns the public key of the message requester and the encryption tag to use, as well as the attribute that it is queried about. Nothing else needs to be included in the policy enforcer's query except a semantically secure ciphertext c (in the case of Figure 18). In reply, the CA returns a ciphertext to the policy enforcer that is equivalent to a fresh encryption. The policy enforcer can learn nothing from this ciphertext by the semantic security of the cryptosystem. □

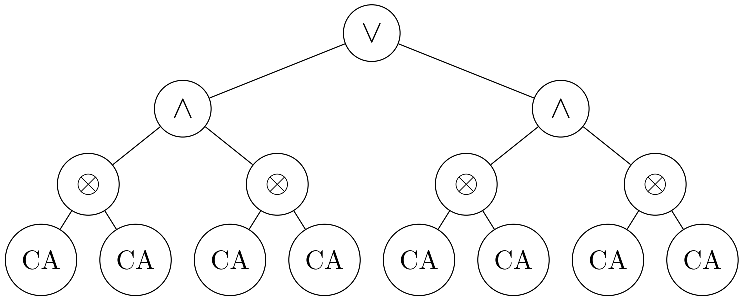

Figure 19: Scrambled circuit with partial homomorphic combination.

5.4 PARTIAL CONJUNCTION EVALUATION

The protocol of the preceding section for evaluating a conjunction to yield a scrambled circuit input wire value requires CAs to be queried sequentially. Although network connection setup and some processing can happen in parallel, each CA in the conjunction adds one round-trip time of network delay. Over a wide-area network, the accumulated cost of this can be substantial for large conjunctions.

We therefore point out that there is a latency/bandwidth trade-off involved in applying homomorphic conjunction evaluation instead of using AND gates in the scrambled circuit. Since the output of the conjunction evaluation protocol is only needed at the time of circuit evaluation, the circuit can be generated and transmitted to the message requester by the policy enforcer while the conjunction evaluation protocol is being run with the online CAs (possibly for multiple conjunctions in parallel). If scrambled circuit generation and transmission is faster than conjunction evaluation, then the overall response time may be reduced by moving some of the conjunction evaluation into the scrambled circuit. For example, a large conjunction can be split into two or more parts, and the Boolean result of evaluating each part can be provided as input to AND gates in the circuit. This creates a larger circuit, but reduces the size of the largest conjunction to be evaluated homomorphically by half with each layer of AND gates added to the bottom of the circuit. Such a situation is illustrated in Figure 19.

5.5 IMPLEMENTATION AND PERFORMANCE EVALUATION

As in Chapters 3 and 4, experiments were conducted using Amazon EC2 virtual machine instances of the "High-CPU Medium Instance" type, provisioned with two cores of a 2.33 GHz Intel Xeon E5410 CPU and 1.7 GB of memory and running the 32-bit Ubuntu 10.04 server distribution. MacKenzie, Reiter, and Yang's tag-based variant of the Cramer-Shoup encryption scheme (Section 5.1) was implemented in C using the GMP library for arithmetic on a subgroup of order q in $\mathbb{Z}_p^*$, where q is a 160-bit prime and p is a 1024-bit prime. The conjunction evaluation protocol of Section 5.3 was implemented in Java, with the cryptographic operations implemented in C invoked through the Java Native Interface (JNI).

Figures 20 and 21 show the time required to evaluate conjunctions of varying size over a local-area network and a wide-area network (as an average of 30 trials). The error bars show 90% confidence intervals for the mean. In the LAN experiment, all instances were located in the US East region and could typically communicate with one another with a round-trip time of less than one millisecond. In the WAN experiment, the policy enforcer was placed in the US East region while the CAs were all located in the Asia Pacific (Singapore) region, with an average round-trip time of about 270–290 ms between the two regions. Much of the cost of evaluating a conjunction of size 1 (querying a single CA) is due to setting up an SSL connection and Java object serialization streams. This is the line labeled "connection setup" in the figures. For conjunctions of size greater than 1, the response time increases linearly with the number of CAs that need to be contacted.

Scrambled circuit generation and evaluation was implemented in Java using the SHA-1 hash of two gate input wire values as a one-time pad to encrypt the corresponding gate output wire value. This was benchmarked on an Amazon EC2 instance using a single 2.33 GHz Intel Xeon CPU core and Java SE 1.6.0 update 20. Table 2 shows the average generation and evaluation times over 30 trials for a scrambled circuit consisting of a complete binary tree of gates. The gate type is unimportant, since essentially the same operations are carried out for any binary gate. Circuit generation is about four times slower than circuit evaluation because four truth table entries need to be encrypted for each gate during circuit generation, whereas only one needs to be decrypted during circuit evaluation.

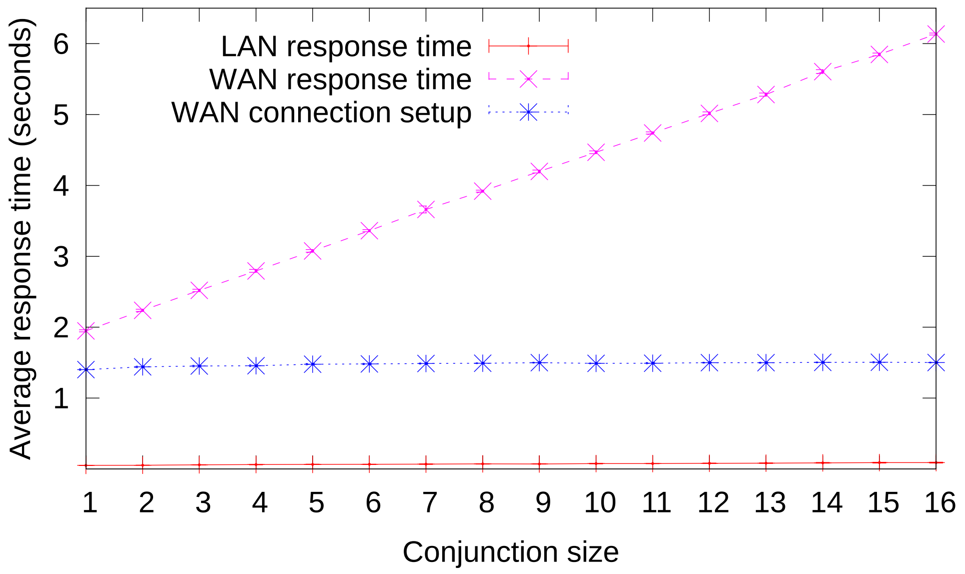

Figure 20: Conjunction evaluation times (average of 30 runs).

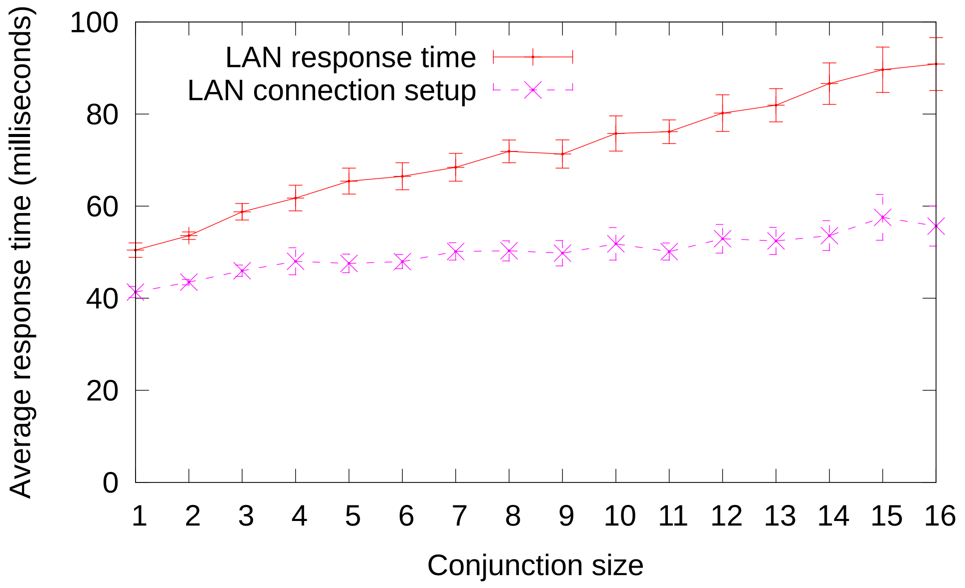

Figure 21: Conjunction evaluation times, showing LAN results alone (average of 30 runs).

Table 2: Scrambled circuit generation, evaluation, and transmission times (ms).

Number of gates	Generation	Evaluation	Transmission (16 Mbps)
7	0.124	0.026	0.224
15	0.214	0.044	0.480
31	0.394	0.083	0.992
63	0.809	0.174	2.016
127	1.614	0.339	4.064
255	3.205	0.686	8.160
511	6.520	1.357	16.352
1023	13.236	2.830	32.736
2047	28.106	6.244	65.504
4095	64.709	15.431	131.040
8191	143.364	33.289	262.112
16383	324.611	67.512	524.256
32767	744.880	156.900	1048.544
65535	1460.947	306.816	2097.120

The circuit can be transmitted over the network to the circuit evaluator as it is generated. If wire values are 128 bits, then each binary gate is 512 bits in size, since the truth table consists of four encrypted values. Thus a 1 Mbps network connection can transmit only 1,953 gates per second. This means that circuit transmission time is likely to be higher than circuit generation time. For example, this is the case in Table 2, which compares circuit generation and transmission times assuming a circuit size of 512 bits per gate and a 16 Mbps network connection. The cost of transmitting circuit layout information is negligible for a binary tree.

Two decryption operations of the homomorphic cryptosystem are needed in order for the message requester to obtain a circuit input wire value from the output of the conjunction evaluation protocol. This cost is not shown in Table 2. On the machines used in these experiments, one CPU core was capable of over 500 decryptions per second in 32-bit mode using GMP 4.3.2. The size of each ciphertext, which must be transmitted to the message requester along with the scrambled circuit, is 4096 bits.

5.6 DISCUSSION

In comparison with FAL, the protocols of this chapter have the disadvantage that online CAs learn which attributes of which requesters they are queried about, revealing partial information about the policies of resource owners and the resources that are requested by users. The gain in policy privacy achieved by not having users submit credentials is substantial, however. Over many interactions, the access requester in FAL can gradually reduce the set of credentials submitted until one or more minimal satisfying sets are found. This is impossible when all credentials come from online CAs, since the access requester simply learns whether access is granted. Note also that the policy enforcer cannot determine a requester's attributes by varying the policy over time, since the enforcer never learns the result of the access control decision.

The computation required of the policy enforcer by the protocols of this chapter is essentially reduced to scrambled circuit generation. Although the protocol response time from the point of view of the access requester is heavily influenced by the network latency incurred in contacting online CAs, the light computational load placed on resource owners compares very favorably with FAL for servers that must be able to handle multiple clients simultaneously. Furthermore, in the protocols of this chapter the scrambled circuit for any policy with a given gate structure can be precomputed in advance while a server is idle. This is also possible in FAL, but in that protocol most of the computational cost for the policy enforcer is private polynomial evaluation and oblivious transfer, which must be done online at the time of the access request.

As noted previously, the option of evaluating conjunctions by homomorphically combining encrypted responses from CAs is a performance optimization that may be applied selectively to individual pairs of attributes. In general, querying CAs sequentially is acceptable when they are "nearby" in terms of network latency (or response time). Any CAs that take longer to respond should be excluded from sequential conjunction evaluation, and their responses can be integrated into a policy conjunction later using AND gates in the scrambled circuit evaluated by the access requester. Keeping historical data on the response times of different CAs naturally facilitates this optimization, just as relational database management systems maintain auxiliary information that aids in query optimization.

Response times for the access requester can be reduced further by pipelining the generation and evaluation of the scrambled circuit. That is, the policy enforcer can send parts of the circuit to the requester as soon as they are generated, along with the encrypted circuit input wire values as they are received from online CAs. The access requester can then immediately evaluate any part of the circuit for which the corresponding input wire values are available, even while waiting for additional encrypted gates and wire values.

An evaluation of the effectiveness of the optimizations mentioned above is left for future work. We also recognize that it is desirable to study the cost of evaluating realistic policies. While lack of real-world policy data is a pervasive problem in the research literature on access control, we hope to develop benchmark scenarios that can be used to empirically compare the different approaches studied in this thesis.

Finally, this chapter has assumed that the CAs, by encrypting attribute values for the access requester's public key, are willing to reveal all attribute values to a requester colluding with the policy enforcer. If this is not the case, then a CA may be willing to conditionally reveal an attribute value to the requester only if the requester has some other attribute that is known only to another CA. The homomorphic conjunction evaluation protocol of Section 5.3 supports these "recursive" access control policies.

First consider the part of the protocol in which a CA returns the encryption of 1 for a true attribute and the encryption of a random value for a false attribute. If the CA plays the part of the policy enforcer by requesting additional encrypted attribute values from other CAs and homomorphically combining them with its own response before returning it to the original policy enforcer, then the final result is the encryption of 1 only if all of the requested attribute values were true. Otherwise, if one of the CAs queried recursively returns the encryption of a random value, this random value completely hides the response of the CA queried by the original policy enforcer, since it is homomorphically multiplied by a random value. Thus any CA can expand the conjunction that is being evaluated by adding other attributes.

An analogous operation is possible for the sequential part of the conjunction evaluation protocol that results in the encryption of 1 when the conjunction is false. A queried CA (at any level of recursion) can act as the policy enforcer by sending the ciphertext being passed along sequentially from CA to CA to additional CAs of its own choosing. Any one of these CAs (and any CAs queried recursively by them) will then have the opportunity to replace the ciphertext with the encryption of 1 if its attribute is false, or leave the plaintext value unchanged while adding new randomness to the encryption if its attribute is true.

6.0 CONCLUSIONS

One of the great weaknesses of Internet security today is the widespread use of password-based authentication, as many users inevitably choose easy-to-guess passwords or reuse the same password for multiple sites. Public key cryptography enables stronger authentication using identity certificates issued by trusted certification authorities (CAs), but any access control mechanism based on proofs of identity requires a resource owner to know the exact identities of everyone who should be given access. In many cases, however, an access control policy is more logically specified in terms of the attributes of authorized individuals. For example, a certain Web page should only be viewable by graduate students in a university's computer science department, or only by students enrolled in a particular course. Attribute-based access control refers to policies like these that make the access control decision a function of attributes other than identity.

A straightforward implementation of attribute-based access control is to have CAs issue certificates containing verified attributes of the subject, just as they issue certificates containing identity information today. These "digital credentials," analogous to physical credentials such as a driver's license or a student ID card, can be used by the credential holder to prove that an attribute-based access control policy is satisfied. From the point of view of security, this is a satisfactory solution as long as the CAs are trustworthy. From the point of view of privacy, however, two major problems arise.

First, credential holders must weigh the benefits that can be gained by the use of their credentials against the loss of privacy entailed by revealing their attribute values. Second, credential holders must know what attributes they must prove in order to satisfy an access control policy. Asking clients to disclose all of their credentials is usually unrealistic, but making the policy public can reveal information about the protected resource.

Ideally, the party enforcing an access control policy learns nothing about the attributes of the party requesting access, and the party requesting access learns nothing about the access control policy besides whether it was satisfied. Since the policy enforcer cannot be allowed to learn anything about the requester's attributes, the enforcer must not even learn whether the policy was satisfied. In the solutions studied in this thesis, the policy enforcer consults online CAs to obtain information about an access requester's attributes. This information is obscured or encrypted in such a way that the policy enforcer does not learn the actual attribute values, but the CAs' replies still remain useful for policy evaluation, which also produces an encrypted result.

Two main approaches were considered. First, a CA can share an attribute value with other CAs using Shamir secret sharing. A specified number of shares (the threshold) is needed to reconstruct the secret, so as long as enough of the CAs are honest, attribute values remain private. Then an arithmetic circuit representing the policy is jointly evaluated by all the protocol participants, who each hold a Shamir share of the value of each wire. This circuit's output wire value is the requested message if the policy is satisfied, and is 0 otherwise. Shares of this output wire value are encrypted for the message requester's public key in an additively homomorphic cryptosystem, and then the reconstruction of the actual circuit output takes place obliviously by homomorphic operations on the ciphertexts. The main problems with this approach are that the arithmetic circuit representing the policy is disclosed to all CAs, and that CAs must trust a majority of the protocol participants.

The second paradigm considered in this thesis is for CAs to encrypt requested attribute values directly for the message requester's public key in a multiplicatively homomorphic cryptosystem. The policy enforcer homomorphically combines the encrypted attributes in a conjunction to obtain the encrypted truth value of the conjunction, and then homomorphically combines this with an encryption of the requested message. The message is left unchanged if the conjunction was true, while it is hidden through multiplication by a random value otherwise. This protocol can be run once for each conjunction in a DNF policy, revealing the number of conjunctions that proved to be true.

As an alternative, the disjunctions in the policy can be evaluated using a scrambled circuit. This reveals only an upper bound on the number of disjuncts. In order to compute a predictable value for both the cases when a conjunction is true and when it is false, the conjunction evaluation protocol must be modified to query CAs sequentially (in the previous variant, CAs could be queried in parallel). The network latency incurred in sequentially querying CAs can be reduced, however, by only partially evaluating the conjunctions in the scrambled circuit, at the cost of a larger circuit size.

Implementations of the previous state-of-the-art solution (Frikken, Atallah, and Li's protocol) and the proposed schemes using online CAs demonstrate that consulting online CAs can improve performance by an order of magnitude. A disadvantage of the online CA approach, however, is that the CAs now learn which credentials are requested, by whom, and when, whereas the FAL protocol uses cryptographic credentials that do not require interaction with the credential issuer. A fundamental privacy/performance trade-off therefore remains between an access requester trying a large number of credentials because the access control policy is hidden, and allowing the policy enforcer to request encrypted attribute values from online CAs. Since both the hidden credentials approach of the FAL protocol and the online CA approach studied in this thesis can be used to obtain the values of input wires to a scrambled circuit evaluated by the access requester, the two alternatives are not mutually exclusive.

BIBLIOGRAPHY

[1] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Comparable algorithm strengths. In *NIST Special Publication 800-57: Recommendation for Key Management*, pages 61–64. U.S. National Institute of Standards and Technology, Mar 2007.

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

[3] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, pages 48–63, 1998.

[4] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[5] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Proceedings of EUROCRYPT 2000*, pages 431–444, 2000.

[6] R. W. Bradshaw, J. E. Holt, and K. E. Seamons. Concealing complex policies with hidden credentials. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 146–157, 2004.

[7] M. Chase. Multi-authority attribute based encryption. In *Proceedings of the Fourth Theory of Cryptography Conference*, pages 515–534, 2007.

[8] M. Chase and S. S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 121–130, 2009.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. Internet Engineering Task Force RFC 5280, May 2008.

[10] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proceedings of CRYPTO 1998*, pages 13–25, 1998.

[11] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Proceedings of ASIACRYPT 2002*, pages 77–85, 2002.

[12] S. Farrell, R. Housley, and S. Turner. An Internet attribute certificate profile for authorization. Internet Engineering Task Force RFC 5755, Jan. 2010.

[13] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of EUROCRYPT 2004*, pages 1–19, 2004.

[14] K. Frikken, M. Atallah, and J. Li. Hidden access control policies with hidden credentials. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pages 27–28, 2004.

[15] K. Frikken, M. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Transactions on Computers*, 55(10):1259–1270, 2006.

[16] M. Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, 2010.

[17] O. Goldreich. *Foundations of Cryptography*, volume 2, chapter 7. Cambridge University Press, 2004.

[18] J. E. Holt, R. W. Bradshaw, K. E. Seamons, and H. Orman. Hidden credentials. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, pages 1–8, 2003.

[19] A. Ivan and Y. Dodis. Proxy cryptography revisited. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, 2003.

[20] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[21] J. Li and N. Li. A construction for general and efficient oblivious commitment based envelope protocols. In *Proceedings of the 8th International Conference on Information and Communications Security*, pages 122–138, 2006.

[22] J. Li and N. Li. OACerts: Oblivious attribute certificates. *IEEE Transactions on Dependable and Secure Computing*, 3(4):340–352, 2006.

[23] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. *Distributed Computing*, 17(4):293–302, 2005.

[24] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[25] P. MacKenzie, M. K. Reiter, and K. Yang. Alternatives to non-malleability: Definitions, constructions, and applications. In *Proceedings of the First Theory of Cryptography Conference*, pages 171–190, 2004.

[26] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay: A secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[27] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, 2001.

[28] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of EUROCRYPT 1999*, pages 223–238, 1999.

[29] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[30] B. Wongchaowart and A. J. Lee. Oblivious enforcement of hidden information release policies. In *Proceedings of the Fifth ACM Symposium on Information, Computer, and Communications Security*, pages 324–327, 2010.