Concealing Complex Policies with Hidden Credentials [*]

Robert W. Bradshaw, Jason E. Holt and Kent E. Seamons
Internet Security Research Lab
Brigham Young University
rwb43@email.byu.edu, isrl@lunkwill.org, seamons@cs.byu.edu

ABSTRACT

Hidden credentials are useful in protecting sensitive resource requests, resources, policies, and credentials. We propose a significant performance improvement when implementing hidden credentials using Boneh/Franklin Identity Based Encryption. We also propose a substantially improved secret splitting scheme for enforcing complex policies, and show how it improves concealment of policies from nonsatisfying recipients.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Authentication

General Terms

Algorithms, Security, Theory

Keywords

Authentication, Privacy, Credentials, Trust Negotiation, Secret Sharing, Identity Based Encryption

1. INTRODUCTION

If Alice and Bob share a secret, they can use it as an authentication token or digital credential. Alice can use the secret to encrypt resources and send them to Bob over insecure channels, and unauthorized parties will gain nothing from the ciphertext. Public key cryptography extends this scenario to situations when Alice and Bob have different credentials—or when Alice has no credentials at all—using digital certificates signed by trusted third parties. Bob shows his certificates to Alice, Alice verifies that the certified attributes satisfy her policy for releasing the resource, and Alice encrypts her resource using Bob's public key. That is, Alice can send messages to Bob based on possession of credentials which she cannot issue: she knows neither the credential issuer's private key nor Bob's private key.

Hidden credentials extend the situation even further. Using Identity Based Encryption (IBE), Alice can create a public key corresponding to an arbitrary string and the public key of a trusted third party. Only the trusted third party can issue the corresponding private key to the "owner" of the string. Hidden credentials leverage this by encoding attributes in strings according to a template published by the trusted third party. For instance, the FBI might publish a template "(nym):FBI agent:(current year)" for FBI agent credentials which expire at the end of each year, and issue Bob the private key corresponding to "Bob:FBI agent:2004". Alice can then send messages to Bob based on credentials which he may or may not have. She may not know that Bob is an FBI agent, but she knows that if he is, he must know the private key corresponding to "Bob:FBI agent:2004". Using secret splitting, Alice can create messages which require any number of credentials for decryption. Consequently, she can encrypt a resource in such a way that only Bob can decrypt the resource, and only if he has credentials sufficient to satisfy Alice's access policy. Bob can do this without Alice's help, with the result that Alice never needs to learn what credentials Bob has.

This has interesting consequences.

First, hidden credentials can solve the "going first" problem in PKI-based authentication systems. Normally, if Alice and Bob wish to establish a trust relationship, one of them must volunteer to go first, showing a credential to a stranger about whom he or she knows nothing. Hidden credentials allow Alice to enforce her policies without having to see Bob's credentials, and vice versa. This also means that combinations of policies which have unresolvable dependency cycles in traditional trust negotiation work just fine with hidden credentials.

Second, policies can be concealed from unauthorized recipients. In many cases disclosure of a policy which requires a sensitive credential is a red flag to attackers that the resource it protects is valuable. Hidden credentials use secret splitting schemes which conceal policy contents from unauthorized parties; only a recipient who holds the credential required by a policy learns that the credential is involved. The improved secret splitting scheme we present here limits this partial disclosure even further, so that only recipients who fulfill complete subexpressions of a policy can determine that they have done so.

[*] This research was supported by funding from DARPA through SSC-SD grant number N66001-01-1-8908, the National Science Foundation under grant no. CCR-0325951 and prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

CCS'04, October 25-29, 2004, Washington, DC, USA.
Copyright 2004 ACM 1-58113-961-6/04/0010 ...$5.00.

Third, it means that credentials can be created and used which are so sensitive that they're never shown to anyone. If Alice is a whistleblower and suspects that coworker Bob is actually an undercover investigator, she can send him evidence encrypted against the FBI credential he must possess if he really is an investigator. Bob can decrypt the information without blowing his cover, even to Alice.

Fourth, hidden credentials can improve protocol performance, since Alice and Bob don't have to send credentials or policies over the network. Instead, Alice uses Bob's nym to derive the public keys Bob must hold if he is to fulfill her access control policy. The nym may be something she already knows, such as a domain name or IP address, or something she doesn't, like a real name or one-time pseudonym. In many cases, the progressive multi-round policy and credential exchanges used in traditional trust negotiation can be reduced to a single exchange of messages with hidden credentials, since all the policies governing a resource can be enforced in a single ciphertext.

The contributions of this paper are as follows:

- We show how to speed up hidden credential decryption operations by an order of magnitude when using the Franklin/Boneh IBE from [6], and show how these results are reflected in our implementation.

- We present an improved secret splitting scheme. The original scheme revealed information about the operators in the access structure. Our improved scheme further limits what the recipient of a message can learn about the sender's policy.

- We present the notion of policy concealment and show how it is partially provided by our improved secret splitting scheme.

Note that this paper deals only with improving the *Encrypt* function in a hidden credential system, which is used to create ciphertexts which can only be decrypted using the appropriate credentials. [8] describes how such ciphertexts can be used in a variety of protocols to realize features found only in hidden credentials.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 provides basic definitions for hidden credentials. In Section 4, we discuss an approach to achieving an order of magnitude performance improvement when performing multiple secret share decryptions. Section 5 introduces a secret splitting scheme that limits what a recipient can learn about the sender's complex policy. Sections 6 presents the performance results of an implementation of the secret splitting scheme. Section 7 contains conclusions and discussion of our future plans.

2. RELATED WORK

Holt et al. introduced hidden credentials in [8]. They gave a formal description for hidden credentials, including the concept of credential indistinguishability, and showed how to build them using the Franklin/Boneh IBE. Their work also gives compelling examples of the utility of hidden credentials.

Sending Bob a message which he can only read if he has a certain attribute is easy if all people who have that attribute share a common secret. For example, Bob could distribute an RSA private key to all the members of "Bob's Club." Alice could then send messages to club members without being a member herself. However, widely shared secrets tend to leak. Furthermore, since RSA public key encryption has no forward secrecy, all messages sent to club members become vulnerable if the private key is compromised by any one member. Hidden credentials avoid both of these drawbacks by allowing Alice to use a public key which depends upon Bob's attribute *and his identity*. If Bob's secret is compromised, only messages to Bob become vulnerable.

Hidden credentials are built upon identity-based encryption (IBE); attribute values are incorporated into the identity, and the credential issuer's public key is the PKG public key. In [6], Boneh and Franklin describe a suitable IBE, and in section 1.1.2 (Delegation) describe a simple system which includes attributes with identity.

Our work also draws much from the paradigm of trust negotiation [11, 5, 12, 4]. Trust negotiation is based on the idea that sensitive resources and information can be guarded by attribute-based policies that can be fulfilled by publicly verifiable digital credentials issued by some third party. As with trust negotiation, users of hidden credentials set up policies and are granted credentials with which they may obtain access to sensitive resources. Unlike trust negotiation, however, sensitive credentials and policies need not be revealed to be used in obtaining these resources.

In 2003, Balfanz et al. [1] proposed a construct called Secret Handshakes. In their system, Alice and Bob receive pseudonyms from a central authority along with a corresponding secret. These form a credential. Alice and Bob must mutually authenticate, satisfying both that each has received a credential from the same authority. Furthermore, the authority can encode roles into the credential which can be made part of the authentication process. For instance, Alice can verify that Bob is a policeman, but only if she has a driver's license. Secret Handshakes are built from pairing-based key agreements.

Secret Handshakes require Alice and Bob to mutually authenticate using credentials from the same issuer. In contrast, hidden credentials allow Alice to send Bob a message depending only on Bob's credentials—Alice need not even have any credentials of her own. Hidden credentials also allow messages to be encrypted according to complex policies, possibly involving multiple credentials from diverse issuers. In fact, with hidden credentials, even the issuer(s) involved need not be overtly disclosed. This may be useful if association with certain organizations is considered sensitive.

Li, Du, and Boneh describe Oblivious Signature-Based Envelopes (OSBE) [9] as a means to resolve circular dependencies in automated trust negotiation. OSBEs are similar to hidden credentials in that the ability to read a message is contingent on having been issued the required secret. Because OSBEs reveal the contents of the credential, they have the advantage of allowing more general compliance checking, such as checking credential chains. OSBEs also can be implemented using existing X.509 certificates with RSA signatures. However, OSBEs require that Alice and Bob agree on the signature Bob needs to have to decrypt Alice's message. In other words, Alice needs to disclose her policy to Bob, who must then reveal his entire credential, minus the signature, before they can proceed. Hidden credentials avoid this, allowing Alice to send Bob a message without disclosing what credential he must use to decrypt it. This can be very significant if the policies or credential in question are extremely sensitive. In situations where credentials and policies are not sensitive, however, hidden credential users can provide the contents of their policies and credentials, resulting in behavior similar to that of OSBEs.

Prior work in trust negotiation has explored the issue of policy sensitivity. Bonatti and Samarati [5] proposed a framework for regulating service access and information release on the Web. Sensitive requirements are not disclosed to strangers, but must first be satisfied before a transaction can progress. Seamons et al. [10] introduced policy graphs to safeguard sensitive policies from unauthorized access. Yu and Winslett [13] proposed a unified scheme to treat policies as first-class objects and protect them like any other sensitive resource. Hidden credentials are another approach that can be used to safeguard sensitive policy information. Some of these approaches assume that sensitive policies are gradually disclosed over multiple rounds of negotiation. Hidden credentials allow the entire procedure to be condensed in a single message. All previous approaches sometimes require a stranger to submit or use a credential without absolute knowledge that the credential is relevant to the sensitive policy that has not been disclosed. This requirement is mitigated if the stranger has only a few credentials. Sometimes, the context of the interaction will focus the negotiation on just the few relevant credentials. However, with hidden credentials, there is no loss of privacy in trying all relevant credentials.

3. DEFINITIONS

Simple Policy A simple policy is a pair $(attr, Pub)$ where $attr$ is a set of one or more attributes (not including identity) and Pub is the public key of the credential authority (CA) needed to verify those attributes. Note that this makes (“*member*”, Pub_{NRA}) and (“*member*”, Pub_{NSA}) two distinct policies. The fact that one owns or requires a credential issued by a specific CA may be as sensitive as the attributes contained therein. Consequently, credential indistinguishability (defined below) requires protection of both the attributes and CA public keys specified by policies.

The exact form of $attr$ should be well specified so that anyone can easily generate the unique representation of the desired attributes. That way, senders don’t have to inform recipients of their policies, and recipients don’t have to reveal the attributes expressed in their credentials. In some cases, though, it may be desirable for the intended recipient of a message to specify the attributes expressed in his credentials, for example when a sender’s policy requires that a “State” attribute be represented as an element of a set of bit strings {“California”, “Florida”, “Ohio”, ...}. Rather than constructing a complex policy consisting of the OR of all acceptable strings, it may be preferable for the recipient to specify his attribute to expedite the transaction. In this case, hidden credentials become largely isomorphic to traditional credential systems, and can make use of X.509, XML or other types of accepted certificate formats.

Complex Policy A complex policy is a monotonic Boolean expression of one or more simple policies which must be satisfied to decrypt a resource. The form a complex policy takes is determined by the secret splitting scheme selected for the implementation. The original scheme proposed in [8] allows policies to be expressed as monotonic Boolean functions as well as MofN threshold operations. The specification of our improved scheme only includes monotonic Boolean functions, but could be extended to include secret shares from other schemes as the original scheme did.

Credential In our system a credential is a tuple $(nym, attr, Pub, sig)$ where nym is the (pseudo-)identity of the credential holder. $(attr, Pub)$ form a simple policy, and sig is the signature on both $attr$ and nym made with the secret key corresponding to the public key Pub. That is, the CA issues a credential asserting that nym has attribute $attr$ by providing sig to the owner of nym. Note that sig is the only thing which is known only to the credential holder (and CA), and must be kept secret.

Credential Indistinguishability A Hidden Credential System has *credential indistinguishablity* if and only if a recipient can only determine which policy was used to encrypt a message if he fulfills that policy. Specifically, credential indistinguishability implies that no polynomial-time bounded adversary $\mathcal{A}$ has a non-negligible advantage in winning the following game against a challenger: $\mathcal{A}$ makes any number of requests for a randomly generated CA public key or private key corresponding to an identity of his choosing for a total of t requests. $\mathcal{A}$ then chooses an identity nym and two simple policies $\mathcal{P}_0$ and $\mathcal{P}_1$ for which he has not received the corresponding credentials. He sends these two policies, along with nym and a message M of his choosing, to the challenger. The challenger chooses a random bit $b \in \{0, 1\}$ and encrypts M against $\mathcal{P}_b$ and nym. He returns the resulting ciphertext to $\mathcal{A}$. $\mathcal{A}$ makes any number of requests for additional CA keys or private keys for identities other than the challenge identity, then eventually outputs a guess $b' \in \{0, 1\}$ for b. $\mathcal{A}$ wins if $b' = b$. A hidden credential system is said to have credential indistinguishablity if $|Pr[b' = b] - \frac{1}{2}| < 1/f(t)$ for any polynomial $f(t)$.

Policy Indistinguishability Credential indistinguishability can extend to complex policies as well. We define a system to have *full policy indistinguishability* if a polynomial-time bound adversary is unable to gain a non-negligible advantage in winning the above game for any two complex policies $\mathcal{P}_0$ and $\mathcal{P}_1$ for which he does not possess a complete satisfying set of credentials. *Partial policy indistinguishability* asserts that such an adversary is unable to gain a non-negligible advantage in the above game for any two policies for which he has no relevant credentials, but does not guarantee an advantage cannot be obtained if the adversary partially fulfills one or the other of them. Note that each of these properties implies credential indistinguishability as well, since all simple policies are also complex policies.

Weak policy indistinguishability means that a system lacks even partial policy indistinguishability. An example of a system with weak policy indistinguishability is the original secret splitting scheme, whose shares leak information about the Boolean expressions they represent. Our improved scheme provides at least partial policy indistinguishability. There are several anonymous secret splitting schemes which might be able to provide full policy indistinguishability, but these tend to involve much more overhead and are left as an avenue for future research.

We note that to achieve more than partial policy indistinguishability, the correct decryption of a secret share according to a simple policy must be indistinguishable from decryptions performed with incorrect credentials. Otherwise, any secret splitting scheme used to enforce complex policies will necessarily reveal any credentials used in the access structure to recipients who possess those credentials, even if those credentials are insufficient to satisfy the policy, since the recipient will be able to recognize which credentials he used in producing correct decryptions of secret shares.

Hidden Credential System A hidden credential system consists of the four algorithms defined below. Optionally, a setup phase precedes these in cases where a set of system parameters needs to be conventionally agreed upon to ensure indistinguishability between credential authorities.

- Create CA

  To create a Credential Authority, generate a private key and publish the corresponding public key. CAs can be created at any time.

- Issue

  Create a credential certifying that the user identified by *nym* possesses the attribute(s) designated in *attr*.

- Encrypt

  Encrypt a message guarded by a policy $\mathcal{P}$ with a specific intended recipient identified by *nym*, and return the ciphertext. Encrypt should be secure against chosen ciphertext attacks and must provide credential indistinguishability as defined above.

- Decrypt

  Attempts decryption of a ciphertext, returning the plaintext if and only if the set of available credentials issued with respect to *nym* is sufficient to satisfy $\mathcal{P}$.

4. IMPROVING PERFORMANCE

Holt et al. [8] describe a hidden credential system built from the FullIdent Identity Based Encryption system in [6]. In this section, we describe how to achieve an order of magnitude improvement in performance when using policies involving multiple credentials. First we specify our modified system. Then we prove in section 4.1 that individual secret shares are secure. Section 4.2 shows that ciphertexts created by *Encrypt* can only be decrypted by qualified recipients. Section 4.3 then shows how adaptive chosen ciphertext security can be added to *Encrypt*.

BasicIdent, given in [6], is a simplified version of FullIdent which omits integrity protection and thus chosen ciphertext security. We regain chosen ciphertext security by providing message authentication as part of the encryption function. BasicIdent produces ciphertexts of the form $\langle U, V \rangle$, where U is a randomizing value. U is used with a bilinear map and hashed to produce a pad. That pad is XORed with the message to produce V.

We prove that U can be reused across several ciphertexts in BasicIdent. This allows the decryptor to perform a single pairing operation whose result can be reused when attempting to decrypt many ciphertexts. Since pairings dominate the cost of IBE operations in current implementations, this optimization reduces the cost of decrypting multiple ciphertexts from a value linear in the number of credentials to a cost which is nearly constant, as our performance results indicate.

Here we define an implementation of hidden credentials modeled after the original specification and prove that it meets the definition given above.

- Setup

  A public value *params* is conventionally agreed upon for use by all users, almost identical to the values chosen in the Setup phase of BasicIdent. As in the original hidden credential specification, the point P is used by everyone, rather than being chosen independently by each CA. Additionally, we require that one of the hash functions accept a second argument which is used like the counter in CTR mode encryption. We also specify encryption and decryption functions $\mathcal{E}$ and $\mathcal{D}$ for encrypting the actual message to be sent.

  Additionally, we assume a security parameter k and selection of an appropriate secret splitting scheme.

  $$params = \langle q, G_1, G_2, \hat{e}, n, P, H_1, H_2, \mathcal{E}, \mathcal{D} \rangle$$

  where q is a large prime number, G_1 and G_2 are two groups of order q, $\hat{e}$ is an admissible bilinear map from $G_1 \times G_1 \rightarrow G_2$ for which the Bilinear Diffie-Helman Assumption as defined in [6] holds. P is an arbitrary generator in G_1. H_1 and H_2 are cryptographic hash functions such that $H_1 : \{0,1\}^* \times \{0,1\}^* \rightarrow G_1^*$ and $H_2 : G_2 \times \mathbb{Z}^+ \rightarrow \{0,1\}^l$ where l is the length of the shares as defined by the secret sharing algorithm.

  $\mathcal{E}$ is a semantically secure symmetric encryption function with integrity of ciphertexts as defined in [2]. $\mathcal{D}_s(\mathcal{E}_s(m)) = m$ for the corresponding decryption function $\mathcal{D}$, but without explicit knowledge of s it is computationally infeasible to generate a ciphertext W such that $\mathcal{D}_s(W)$ does not fail. In other words, $\mathcal{E}$ is secure against existential forgeries. This can be done by letting $\mathcal{E}_s(m)$ be the encryption of m with a semantically secure symmetric encryption function along with a MAC of the resulting ciphertext that is secure against existential forgeries. $\mathcal{D}$ returns the corresponding decryption if the MAC is correct, and returns failure if it is not.

  Note that H_1 must be collision resistant across both its variables. For instance, H_1(“*ab*”, “*c*”) must not produce the same value as H_1(“*a*”, “*bc*”), since it would then be possible to find multiple $(nym, attr)$ pairs that would produce the same keys, encryptions, and decryptions.

- Create CA

  Each CA chooses a random private key $\alpha \in \mathbb{Z}_q^*$ and publishes its public key $Pub = \alpha P$. CAs can be created at any time.

- Issue

  A CA issues a credential fulfilling a policy $(attr, Pub)$ by computing $sig = \alpha H_1(nym, attr)$ where nym is the (pseudo)nym of the intended credential holder.

  A CA issues a credential fulfilling a policy $(attr, Pub)$ by computing $sig = \alpha H_1(nym, attr)$ where nym is the (pseudo)nym of the intended credential holder.

- Encrypt

  To encrypt a message M for a recipient designated by nym using the policy $\mathcal{P}$, first generate a random bit string $s \in \{0,1\}^k$ and a random value $r \in \mathbb{Z}_q^*$. Split s according to $\mathcal{P}$ and the secret splitting scheme of choice to create n secret shares $s_1, ..., s_n$, to be encrypted against corresponding simple policies $p_1, ..., p_n$. Each simple policy p_i consists of a pair $(attr_i, Pub_i)$. Construct the ciphertext as follows:

$$C = \langle rP, s_1 \oplus \sigma_1, s_2 \oplus \sigma_2, ..., s_n \oplus \sigma_n, \mathcal{E}_s(M)\rangle$$

where

$$\sigma_i = H_2(\hat{e}(Pub_i, Q_i)^r, i) \text{ and } Q_i = H_1(nym, attr_i)$$

- Decrypt

  Upon receipt of a ciphertext $C = \langle U, V_1, V_2, ..., V_m, W\rangle$, the recipient first finds all possible plaintext shares s_{ij} by calculating

  $$s_{ij} = V_i \oplus H_2(\hat{e}(U, sig_j), i)$$

  for each V_i and each of his credential signatures sig_j. The bilinearity of $\hat{e}$ implies that $\hat{e}(\alpha P, Q)^r = \hat{e}(rP, \alpha Q)$, which allows decryption using the signature instead of knowing r as the sender did. Of course, s_{ij} will be nonsense unless sig_j belongs to the right credential. The secret splitting scheme is left to define how the recipient recognizes correct decryptions, if such decryptions are to be recognized at all.

  If he has a satisfying set of shares in his collection of s_{ij}, he can use these to recover s and attempt $\mathcal{D}_s(W)$ which will indicate successful recovery of the message M.

Implementation note: In practice, it may be possible to replace H_2 with a stream cipher suitable for random number generation keyed with $\hat{e}(U, sig_j)$. Then, rather than running H_2 for each attempt at decrypting a key share, treat $(V_1, ..., V_n)$ as a single ciphertext and generate enough keystream to decrypt the entire concatenation. Encryption likewise requires generating the same keystream for a particular V_i.

4.1 Reusing rP

Here we show that reusing the value rP does not compromise the semantic security of BasicIdent. Consequently, our specification of secret share encryption in the *Encrypt* function is also IND-ID-CPA secure, producing secret shares secure against unqualified recipients, since each secret share V_i is essentially a BasicIdent encryption reusing rP. Section 4.2 shows that if encrypted secret shares are individually secure, messages sent using *Encrypt* are likewise secure.

[6] defines an IND-ID-CPA game for BasicIdent in which an adversary tries to distinguish which of two messages of his choosing was encrypted by a challenger to form a challenge ciphertext. The adversary queries the random oracle H_2 and can make extraction queries which return private keys for identities of his choosing. He submits a challenge identity whose private key he has not received, two equal length messages s_0 and s_1, and receives a challenge ciphertext $C = \langle rP, s_b \oplus H_2(\hat{e}(\alpha P, Q_{id})^r))\rangle$. He is then allowed more extraction and H_2 queries, and ultimately guesses which message was encrypted. [6] shows that if he has any significant advantage over random guessing, the BDH assumption must be false.

We extend this game to show that reuse of rP is safe by adding the ability for the adversary to make reuse queries. He specifies a message and an identity different from the challenge identity, and receives the encryption of that message against that identity using the same rP used in the challenge ciphertext. We show that such queries give the adversary no advantage in winning the unmodified IND-ID-CPA game for BasicIdent. Our first proof shows that reuse of rP is safe for an unmodified BasicIdent as long as each BasicIdent ciphertext uses a different id. But since complex policies may specify the same simple policy (and hence the same identity string) multiple times, we give a second proof which shows how our addition of an index variable to H_2 makes rP reuse possible even with reused identity strings. Consequently, each encrypted share V_i produced by *Encrypt* is secure against chosen plaintext attack despite sharing the value rP with many other shares.

THEOREM 4.1. *Reusing rP does not compromise the semantic security of the BasicIdent system proposed in [6] as long as rP is not reused on the same id. Concretely, suppose there is an IND-ID-CPA adversary $\mathcal{A}$ that has advantage $\epsilon(k)$ against the BasicIdent system when the value rP is reused. Suppose $\mathcal{A}$ makes at most q_E private key extraction queries, q_{H_2} hash queries to H_2, and q_R reuse queries. Then there is an IND-ID-CPA adversary $\mathcal{B}$ with runtime $(time(\mathcal{A}))$ that has an advantage $\epsilon(k)$ against the BasicIdent system making at most $q_{H_2} + q_R$ hash queries and $q_E + q_R$ private key extraction queries.*

PROOF. We construct an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ as follows, mirroring the game defined in [6]:

Setup: Algorithm $\mathcal{B}$ starts by giving $\mathcal{A}$ the BasicIdent system parameters.

Private key extraction queries and H_2 queries: Algorithm $\mathcal{B}$ responds to any valid extraction or H_2 query by passing the query on to the challenger and returning the result.

Reuse queries: $\mathcal{A}$ makes a reuse query by specifying a message s, identity id and a point rP. $\mathcal{B}$ runs an extraction query on id to get αQ_{id}, and returns an encryption of s using rP, taking advantage of the bilinearity of $\hat{e}$:

$$\langle rP, s \oplus H_2(\hat{e}(rP, \alpha Q_{id}))\rangle = \langle rP, s \oplus H_2(\hat{e}(\alpha P, Q_{id})^r)\rangle$$

Phase 1: Algorithm $\mathcal{B}$ responds to extraction and H_2 queries as described above.

Challenge: Once algorithm $\mathcal{A}$ decides that it is done with phase 1, it outputs an identity id_{ch} it wishes to be challenged on for which it has not performed an extraction query, as well as two equal-length messages s_0 and s_1. Algorithm $\mathcal{B}$ gives this same challenge to its challenger and sends $\mathcal{A}$ the challenge ciphertext $C = \langle U, V\rangle = \langle rP, s_b \oplus H_2(\hat{e}(\alpha P, Q_{id})^r)\rangle$, an encryption of s_b where $b \in \{0, 1\}$ is chosen at random by $\mathcal{B}$'s challenger.

Phase 2: Algorithm $\mathcal{B}$ responds to extraction, H_2 and reuse queries as above, with the condition that $id \neq id_{ch}$ for reuse and extraction queries.

Guess: Algorithm $\mathcal{A}$ outputs a guess b' for b. Algorithm $\mathcal{B}$ returns b' as its guess to the challenger and wins if $b' = b$.

It is clear to see that $\mathcal{B}$ has the same advantage as $\mathcal{A}$ in winning the game. Algorithm $\mathcal{B}$ makes exactly one extraction query for each extraction and each reuse query of $\mathcal{A}$, so the total number of extraction queries is $q_E + q_R$ as required. By the same logic the total number of hash queries is $q_{H_2} + q_R$. Boneh and Franklin proved in [6] that in the random oracle model the probability for success of such an attack is negligible under the BDH assumption. □

This shows that reusing rP is safe when using an otherwise unmodified BasicIdent with differing identity strings. Likewise, reusing rP for messages with different CAs is something the adversary can already do, since anyone can create a CA, extract the private key for any identity and create ciphertexts which reuse rP. This has implications for using the Frankiln/Boneh IBE system outside of hidden credentials. For example, when sending an email to multiple recipients, one can use the same rP for everyone.

The remaining change which must be addressed is the addition of the index parameter i to H_2. This may also have applicability outside of hidden credentials; by using a unique nonce for the index parameter, multiple messages can be sent to a single recipient without the need to recalculate either rP or the bilinear map. Note that this addition alone is actually sufficient to allow reuse of rP, but we found it particularly interesting that rP can safely be reused even without the index i.

Theorem 4.2. *Adding an index parameter to the hash function H_2 prevents an attacker from learning any information about shares that were encrypted against the same identity. Concretely, suppose there is an IND-ID-CPA adversary $\mathcal{A}$ that has advantage $\epsilon(k)$ against the BasicIdent system when the value rP is reused with different indexes. Suppose $\mathcal{A}$ makes at most q_E private key extraction queries, q_{H_2} hash queries to H_2, and q_R reuse queries. Then there is an IND-ID-CPA adversary $\mathcal{B}$ with runtime $(time(\mathcal{A}))$ that has advantage $\epsilon(k)$ against the BasicIdent system, making at most $q_E + q_R$ private key extraction queries and $q_{H_2} + q_R$ hash queries.*

Proof. Let $Block_l(x, i)$ denote the ith block of length l of the binary string x. That is, $Block_l(x, i)$ is bits $(i-1) \cdot l + 1$ to $i \cdot l$ of x.

We construct an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ as follows:

Setup: Algorithm $\mathcal{B}$ starts by giving $\mathcal{A}$ the BasicIdent system parameters.

Private key extraction queries: Algorithm $\mathcal{B}$ responds to any valid query by passing the query on to its challenger and returning the result.

H_2 **queries:** Algorithm $\mathcal{B}$ responds to an H_2 query $H_2(X, i)$ by querying its challenger for $H_2(X)$ and returning $Block_l(H_2(X), i)$.

Reuse queries: Algorithm $\mathcal{B}$ responds as follows to a reuse query on a message s, an identity id, a point rP, and an index i, where $i \neq i_{ch}$ if $id = id_{ch}$:

- if $id \neq id_{ch}$, $\mathcal{B}$ runs an extraction query on id to get αQ_{id} and returns $\langle rP, s \oplus Block_l(H_2(\hat{e}(rP, \alpha Q_{id})), i)\rangle$
- if $id = id_{ch}$, $\mathcal{B}$ returns $\langle rP, s \oplus Block_l(V, i)\rangle$

Phase 1: Algorithm $\mathcal{B}$ responds to extraction and H_2 queries as described above.

Challenge: Once algorithm $\mathcal{A}$ decides that it is done with phase 1, it outputs an identity id_{ch} it wishes to be challenged on, as well as two messages s_0 and s_1 and an index i_{ch}. Algorithm $\mathcal{B}$ constructs two messages s'_0 and s'_1 such that $Block_l(s'_0, i_{ch}) = s_0$, $Block_l(s'_1, i_{ch}) = s_1$, and $Block_l(s'_0, i) = Block_l(s'_1, i) = 0^l$ for $i \neq i_{ch}$. He gives s'_0 and s'_1 to the challenger who returns the challenge ciphertext $C = \langle U, V\rangle$, an encryption of s'_b where $b \in \{0, 1\}$ is chosen at random by the challenger. B returns $\langle U, Block_l(V, i_{ch})\rangle$ to $\mathcal{A}$.

Phase 2: Algorithm $\mathcal{B}$ responds to extraction and reuse queries as above, with the condition that $id \neq id_{ch}$ for extraction queries and that $id \neq id_{ch}$ or $i \neq i_{ch}$ for reuse queries.

Guess: Algorithm $\mathcal{A}$ outputs a guess b' for b. Algorithm $\mathcal{B}$ returns b' as its guess to the challenger and wins if $b' = b$.

It is clear to see that $\mathcal{B}$ has the same advantage as $\mathcal{A}$ in winning the game. Algorithm $\mathcal{B}$ makes exactly one extraction query for each extraction query of $\mathcal{A}$, so the total number of extraction queries is $q_E + q_R$ as required. Algorithm $\mathcal{B}$ also makes one H_2 query for each H_2 query of $\mathcal{A}$, and up to one H_2 query for each reuse query, so the total number of H_2 queries by $\mathcal{B}$ is at most $q_{H_2} + q_R$.

What remains to be shown is that the reuse queries returned are the correct ciphertexts.

If $id \neq id_{ch}$, $\hat{e}$ is used as in the last proof to create the appropriate ciphertext.

If $id = id_{ch}$, note that $i \neq i_{ch}$, so we have

$$\begin{aligned} Block_l(V, i) &= Block_l(s'_b \oplus H_2(\hat{e}(\alpha P, Q_{id})^r), i) \\ &= Block_l(s'_b, i) \oplus Block_l(H_2(\hat{e}(\alpha P, Q_{id})^r), i) \\ &= 0 \oplus H_2(\hat{e}(\alpha P, Q_{id})^r, i) \\ &= H_2(\hat{e}(\alpha P, Q_{id})^r, i) \end{aligned}$$

So the returned ciphertext $\langle rP, s \oplus Block_l(V, i)\rangle$ is a valid encryption of s against id using the same rP.

Again, the probability of such an attack succeeding is negligible if the BDH problem is hard. □

4.2 Security of Encrypt

Theorem 4.3. *A recipient $\mathcal{R}$ possessing a satisfying set of credentials for a policy $\mathcal{P}$ can recover M from the ciphertext $C = Encrypt(M, nym, \mathcal{P})$.*

Proof. Let $C = \langle U, V_1, V_2, ..., W\rangle$, where each V_i is an encrypted secret share protecting a master secret s, and $W = \mathcal{E}_s(M)$. Since $\mathcal{R}$ has a satisfying set of credentials, she can decrypt secret shares sufficient to recover s. Then $M = \mathcal{D}_s(M)$. □

Theorem 4.4. *Assuming the BDH problem is hard, the secret splitting scheme is information theoretically secure, and $\mathcal{E}$ is semantically secure, an adversary $\mathcal{A}$ without a satisfying set of credentials for the policy $\mathcal{P}$ cannot recover the message M from the ciphertext $C = Encrypt(M, nym, \mathcal{P})$ with a lower work factor than is required to break BasicIdent or $\mathcal{E}$.*

PROOF. Let $C = \langle U, V_1, V_2, ..., W\rangle$, where each V_i is an encrypted secret share protecting a master secret s, and $W = \mathcal{E}_s(M)$. By theorem 4.2, any reasonable adversary who can recover a secret share for which he does not possess the required credential can also compromise BasicIdent, contradicting the BDH assumption [6]. Therefore, $\mathcal{A}$ cannot recover a satisfying set of secret shares and learns nothing, in an information theoretic sense, about s. Without s, recovering M from W requires compromising $\mathcal{E}$. □

4.3 Adaptive Chosen Ciphertext Security

This section shows how *Encrypt* can be changed so that a complete ciphertext remains secure even in the presence of a decryption oracle.

The BasicIdent system as presented in [6] does not provide chosen ciphertext security. Boneh and Franklin use the Fujiaski-Okamoto transformation from [7] to provide chosen ciphertext security in their FullIdent IBE system. Unfortunately, the way in which the Fujisaki-Okamoto transform is applied reveals the value r to the recipient, making FullIdent unsuitable for use with Hidden Credentials which reuse rP, and possibly allowing him to discover parts of the policy that he does not fulfill.

In our system, adaptive chosen ciphertext security can be added as follows: Rather than encrypting the message with the master secret s, one can encrypt it with $\kappa = H_3(s, V_1, ..., V_n)$ where $H_3 : \mathcal{K} \times \{0,1\}^* \times \cdots \times \{0,1\}^* \rightarrow \mathcal{K}$ is a cryptographic hash function where $\mathcal{K}$ is the keyspace of $\mathcal{E}$. We now provide adaptive chosen ciphertext security by leveraging the integrity protection properties of $\mathcal{E}$. Because $\mathcal{E}$ has integrity of ciphertext, it is computationally infeasible to generate a ciphertext W such that $\mathcal{D}_\kappa(W)$ does not fail without explicit knowledge of κ. Specifically, let the probability that $\mathcal{D}_\kappa(W)$ does not fail for a W constructed without knowledge of the key κ be no greater than $\epsilon(k)$ where $\epsilon(k)$ is negligible in the security parameter k. We assume that $1/|\mathcal{K}|$, where $|\mathcal{K}|$ is the size of the keyspace of $\mathcal{E}$, is also negligible with respect to the security parameter.

THEOREM 4.5. *Giving an attacker access to a decryption oracle does not give an attacker a non-negligible advantage against a challenge ciphertext.*

PROOF. Let $\mathcal{A}$ be an adversary who is being challenged on a ciphertext $C = \langle U, V_1, V_2, ..., V_m, W\rangle$. $\mathcal{A}$ makes a decryption query on the ciphertext

$$C' = \langle U', V_1', V_2', ..., V_{m'}', W'\rangle$$

Suppose $W' = W$. If no master secret is discovered in the decryption of C' then the decryption oracle outputs failure. Otherwise call the recovered master secret s'. Let $\kappa = H_3(s, U, V_1, V_2, ..., V_m)$ and $\kappa' = H_3(s', U', V_1', V_2', ..., V_{m'}')$. Because $\mathcal{A}$ is not allowed to request the decryption of the ciphertext he is being challenged on,

$$\langle U', V_1', V_2', ..., V_{m'}'\rangle \neq \langle U, V_1, V_2, ..., V_m\rangle$$

and hence $\Pr[\kappa' = \kappa] = 1/|\mathcal{K}|$. So if $W' = W$ then $\mathcal{D}_\kappa'(W)$ fails with probability at least $1 - (1/|\mathcal{K}| + \epsilon(k))$.

Suppose $W' \neq W$. Either $W' = \mathcal{E}_\kappa(M)$ for some key κ known to $\mathcal{A}$ (in which case the decryption oracle yields no additional information) or $\mathcal{D}_\kappa(W')$ fails with probability at least $1 - \epsilon(k)$.

In either case, the decryption oracle does not provide any additional information with overwhelming probability. Thus, access to a decryption oracle does not provide a non-negligible advantage against the challenge ciphertext. □

5. SECRET SPLITTING

In hidden credentials, access to a resource is granted based on the satisfaction of a policy defined as an access control structure in a secret splitting scheme. Many general secret splitting schemes have been proposed, and most of them can be used with hidden credentials by creating secret shares that correspond to credentials the recipient may hold, then encrypting each share against its required credential.

5.1 Policy Concealment

Still, care must be taken in choosing a secret splitting scheme in order to maximize policy indistinguishability. The form of an access structure, or even just the number of terms it contains can leak information. For example, consider an attacker named Mallory trying to learn information about a secure database. Each time he sends a request to the database, it returns a response encrypted against some set of hidden credentials chosen according to the database's access policy for that entry. The database doesn't want to reveal which entries exist in the database and which don't, so it always responds to requests for nonexistent entries with a fake response encrypted against credentials which are never issued to anyone ("NAK" credentials).

If we assume that Mallory has no valid hidden credentials, then he should not be able to learn anything about the database, including what entries it contains. But what if the secret splitting scheme implicitly reveals information about the access structure? For example, Mallory might be able to infer, based on the size of the encrypted secret shares returned by the database, that the access policy for a particular resource is $? \wedge (? \vee ?)$. That is, Mallory doesn't learn what credentials correspond to each ?, but he does learn that the policy consists of an AND and an OR. If the database always uses the same access structure when returning responses for nonexistent entries, we have a problem. Mallory simply asks for an entry which he knows is nonexistent, and examines the access structure of the response. Now he knows that any time he makes a request and receives a response with a *different* access structure, that entry must exist in the database, even if he can't actually learn anything about its value.

The following aspects of a secret splitting scheme can all leak information about a transaction:

- The form of the access structure, eg. $? \wedge (? \vee ?)$
- The number of credentials involved in the access structure
- Which credentials possessed by the recipient contribute to partial fulfillment of the access structure. For example, learning that a particularly sensitive credential was necessary but not sufficient to satisfy the access structure suggests that the resource itself is quite sensitive.

A system with full policy indistinguishability would leak no information about any of these aspects to recipients who don't completely satisfy the access structure. As we stated earlier, our improved scheme provides partial policy indistinguishability, whereas the original scheme provided only weak policy indistinguishability.

5.2 Improved Secret Splitting

In the original hidden credential specification [8], encrypting a message M according to a policy $(X \wedge Y) \vee Z$ produces a ciphertext $\{E_X(E_Y(M)), E_Z(M)\}$, where $E_X(M)$ denotes encryption of M according to simple policy X. The Boneh/Franklin IBE system produces ciphertexts longer than the input plaintext, so the recipient can easily discern that the first term in the ciphertext involves a double encryption and therefore has an AND structure. The second term reveals that an OR is present, since ORs always produce additional terms. Consequently, with no decryption effort at all, the recipient knows that the policy used was $(?\wedge?)\vee?$.

Additionally, the original specification assumed that the recipient was provided a way to recognize correct decryptions of elements in the ciphertext. That is, decrypting the first term in the example above with the X credential produces a "correct" decryption, implicitly revealing that X is part of the policy. Of course, without the X credential, the recipient can't attempt decryptions of the inner $E_Y(M)$ even if he has Y. But he can learn about X's presence in the policy if he has it.

Our system improves both of these shortcomings. Rather than receiving a structured ciphertext with elements of various sizes, the recipient gets a number of secret shares, all of equal size. The number of shares places an upper bound on the number of terms in the policy, but not a lower bound since the sender can include any number of bogus shares with only a marginal increase in overhead.

Furthermore, message recipients only learn portions of an unfulfilled policy if they can satisfy a complete subexpression. That is, in a policy of $((W \wedge X) \wedge Y) \vee Z$, the recipient only learns that credentials W and X are in the policy if he possesses *both* of them.

Finally, our system includes ambiguity about partial policy fulfillment. In the above example, if the recipient has credentials W and X, he learns that they are *probably* part of an AND subexpression, but is left with some chance that they were incorrect decryptions which just happened to match up. This type of ambiguity can be hard to preserve across multiple transactions, and so we leave it mostly as an avenue for future research. We treat the topic briefly in the Parameter Selection section below.

Overview. We propose a modification of the simple secret splitting system found in [3]. In that system, if either of secret shares X OR Y is sufficient to recover a secret s, we simply set $X = s$ and $Y = s$. If X AND Y are both required to recover s, we generate a random pad v and set $X = s \oplus v$ and $Y = v$. Recursively applying this procedure allows us to split a secret s into a set of shares $\{s_1, ..., s_m\}$ for any monotonic Boolean access structure. That is, s can be recovered if and only if one possesses a satisfying set of secret shares. Each one of the shares is a bit string of equal length to the original secret.

In our modified system, rather than overtly giving the access structure, we mark the shares at each step. First, we mark the ultimate secret s with a prefix d to indicate when s has been successfully recovered. To split s using an AND structure, we prepend a set number of random bits to each share after applying the random pad. When both shares are recovered, their matching prefixes identify them as operands of an AND which should be XORed together. Splitting a secret with an OR structure produces identical shares, so no prefix is necessary to identify the matching shares.

We also pad all shares on the right to make each have the same length, so that the number of prefixes prepended to a given share doesn't reveal its depth in the access structure. This padding is added to the master secret before splitting. All resulting shares can then be truncated to an equal length.

Definition. Our system is defined in three phases: setup, sharing and recovery. We assume a message sender A who will define access structures and issue secret shares to be encrypted against particular hidden credentials, and a recipient B with a set of credentials $\mathcal{C_B} \subset \mathcal{C}$, where $\mathcal{C}$ is the set of all possible credentials. The set of all simple policies $\mathcal{P}$ is defined as $\bigcup p|(p = (attr, Pub)) \wedge ((nym, attr, Pub, sig) \in \mathcal{C})$. '$||$' denotes concatenation.

Setup:

1. A chooses a security parameter k.
2. A chooses a symmetric encryption function with integrity protection $\mathcal{E}$ and its corresponding decryption function $\mathcal{D}$, such that $m = \mathcal{D}_s(\mathcal{E}_s(m))$. Integrity protection, such as a MAC, is required to allow use of short values of the "done" prefix (defined below) without leaving ambiguity as to whether the secret has been properly recovered, and to ensure that A cannot attack the system by creating shares which appear to recover different secrets depending on how B satisfies the policy. That is, the integrity protection must ensure that A cannot find two secrets s_1 and s_2 which both provide valid decryptions of a ciphertext.

Sharing:

1. A chooses the prefix length l_p
2. A chooses the "done" prefix $d \in \{0,1\}^*$
3. A chooses a secret $s' \in \{0,1\}^k$ to split according to an access structure f. The Encrypt algorithm encrypts her message m to produce $ciphertext = \mathcal{E}_{s'}(m)$.
4. The "done" prefix d is prepended to s' to indicate when s' has been successfully recovered. Then s' is padded to the right with a random value $v \in \{0,1\}^{l_p|\mathcal{S}|}$, where $|\mathcal{S}|$ is the total number of secret shares (including bogus shares) which will be produced. For convenience in this specification we call the chosen secret s' and define $s = d||s'||v$. We call $l = |s|$ the share length, since all shares will be this size.
5. A chooses f, a monotonic Boolean formula defined in terms of credentials in $\mathcal{C}$.
6. f is recursively evaluated to produce a set of shares $\mathcal{S}$, one for each operand in f. A calls $Split(s, f)$, where $Split$ is defined as follows:

   if $f = f_0 \vee f_1$,

   (a) call $Split(s, f_0)$
   (b) call $Split(s, f_1)$.

   if $f = f_0 \wedge f_1$,

   (a) truncate the rightmost l_p bits of padding from s
   (b) choose a random prefix $p \in \{0,1\}^{l_p}$
   (c) choose a random pad $v \in \{0,1\}^{l-l_p}$

(d) call $Split(p||(s \oplus v), f_0)$

(e) call $Split(p||v, f_1)$.

if f is a simple policy,

(a) Add a share to $\mathcal{S}$ along with its simple policy p. $\mathcal{S} := \mathcal{S} \cup (s, p)$. The Encrypt algorithm handles encryption of s against p.

7. A creates a number of bogus shares so that the number of shares in $\mathcal{S}$ doesn't reveal the number of terms in f. Each bogus share is chosen at random as $v \in \{0,1\}^l$, and then (v, NAK) is added to $\mathcal{S}$, where NAK is a policy for a credential that is never issued to anyone.

8. A randomly rearranges the shares in $\mathcal{S}$ so that the position of a share doesn't leak information about f.

9. A sends Encrypt's encryption of $\mathcal{S}$ and M, along with the unencrypted values (l_p, d, k), to B.

Recovery:

1. B creates an empty table $T = \{\}$

2. For each $s_i \in \mathcal{S}$,

   (a) Iterate over the credentials $c_j \in \mathcal{C_B}$ and add each attempt at decrypting c_j with s_i.

   (b) If any two elements $t_a, t_b \in T$ are equal, remove one of them from T. (t_a and t_b are operands of an OR).

   (c) If any two elements $t_a, t_b \in T$ begin with the same prefix, strip the prefix off and add the XOR of the two elements to T. (t_a and t_b are probably operands of an AND). That is, $\forall \{t_a, t_b\} \in T \times T,\ IF\ \exists p \in \{0,1\}^{l_p} | (t_a = p||x) \wedge (t_b = p||y)\ THEN\ T := (T \cup x \oplus y)$.

   (d) If any element $t_a \in T$ has the "done" prefix, then s', where $t_a = d||s'$, is the secret with probability 1 in $2^{|d|}$. If $m = \mathcal{D}_{s'}(ciphertext)$ indicates successful decryption, m has been successfully recovered. Halt.

Once B has recovered a valid secret, he may wish to continue the algorithm rather then halting, to determine if there are any other ways in which the policy might be satisfied. This can be beneficial if he wishes to reveal to A that he successfully recovered s'. For example, if recovering s' required B to use c_1, a very sensitive credential, B may be unwilling to reveal to A that he successfully recovered s', because it implicitly reveals his possession of c_1. On the other hand, if s' could be recovered with either of c_1 or c_2, where c_2 is a non-sensitive credential, then B can respond as if c_2 were the only credential he owned.

5.3 Parameter Selection

Prefix length. One parameter in our system, the prefix length l_p, must be chosen with care to ensure termination of the recovery algorithm. Prefixes indicate whether two shares form arguments to a simple AND subexpression of the access structure. Since incorrect decryptions are assumed to produce essentially random prefixes, the birthday paradox predicts that random prefixes of length n will produce a false match in a set with about $2^{n/2}$ random elements. Such false positives increase storage and computational costs in the recovery phase, but decrease how much a recipient learns about a policy which he only partially satisfies. If the prefix length is set *too* low, recovery can actually become intractable. Consider a prefix of zero length—all terms would "match", producing new terms which would also match all the others, ad infinitum. Share lengths are finite, so this process would terminate eventually for any non-zero prefix length, but would take an exponential amount of time. Our results indicate that 16-bit prefixes provide some ambiguity, but still ensure that, with a probability of less than 2^{-100} for even reasonably large tables, the table will not even double in size as a result of false positives.

In any case, such ambiguity only extends to operands of AND subexpressions, since operands of an OR are identical in all their bits. Expressing the formula as a sum of products reduces the number of OR terms at the terminating level of the tree. It should be noted, however, that no information about either the OR or AND operands is revealed unless one has both of two *corresponding* shares.

It should also be noted that the ambiguity provided by short prefixes is fragile. If a recipient can request multiple ciphertexts encrypted with the same policy, he can quickly separate false positives from the matches which occur every time. The nature of the resource itself and the policies governing related resources can also provide clues as to which credentials are likely to be involved in a particular policy. Future work may treat this idea in more depth and explore ways to further exploit such ambiguity, such as extending it to OR subexpressions.

The "done" prefix. d may have any value, but its length is significant. Incorrect decryptions of secret shares will produce d as a prefix with probability $2^{-|d|}$, so d should be chosen large enough to avoid too many false positives, but small enough that encrypted shares are of reasonable length. The length of d does not impact policy indistinguishability, as an incorrect value of s' is easily identified by the integrity protection in the encryption algorithm.

Bogus shares. The time required to decrypt a message in our system is presently dominated by the number of elliptic curve operations, which in the improved decryption algorithm is only determined by the number of credentials held by the recipient. Consequently, adding bogus shares does not significantly impact the runtime of the decryption algorithm. In our system, encrypted shares are indistinguishable from random bit strings and so encryption doesn't suffer a performance penalty either. We recommend using the same size ciphertext for any message, no matter how simple the policy. Simply find the upper bound for the number of shares over all possible policies (say, the policy that would have to be fulfilled to release all resources), add any number of additional shares to conceal that upper bound, and use that number of total shares for all encryptions. Thus a "NAK" policy used for nonexistent resources, consisting entirely of bluff shares, is indistinguishable from any other system policy to an unqualified recipient.

5.4 Performance

Share length and table size increase with the number of secret shares. Table size also increases with the number of credentials used for decryption. However, since prefix lengths are typically short, on the order of 16 bits, share length increases slowly. For very large policies, share length could even be set to a value lower than the maximal default specified above.

False positives can impact performance in two ways. First, if two shares match coincidentally, an additional share is added to the table which may in turn match with other shares. This can be mitigated by choosing a large enough prefix size. For example, with a prefix size of 16 bits, the chances that n random shares will result in n false positives using this algorithm (effectively doubling the table size) is less than 2^{-100}.

Secondly, false positives can also occur when a share coincidentally starts with the done prefix d. If there are a total of n shares at the termination of the algorithm, on average $n/2^{|d|}$ of them will begin with d. Thus there will be very few false positives of this type for any reasonable length of d. Also, the symmetric key operations involved in attempting decryption will generally be dwarfed by the elliptic curve operations, so the few false positives that occasionally occur will be quickly identified without much overhead. If this is not the case (for example, if the message is in the order of hundreds of megabytes) the length of d can be made large enough to reduce the probability of false positives of this sort as low as desired.

Since the recovery operations consist primarily of XOR and comparison operations, even large tables can be processed absurdly quickly—much more quickly than even a single IBE operation, as our experimental results confirm.

5.5 Security of the Secret Splitting Scheme

Benaloh and Leichter [3] proved that their system provides perfect information hiding. Our system inherits this same perfect information hiding and our proof follows theirs.

LEMMA 5.1. *A recipient B who does not satisfy the chosen access structure f is unable obtain any information about s'.*

PROOF. We prove this by induction on the number of terms in f.

Suppose f has only 1 term. Because B does not satisfy f, s' is encrypted against a credential B does not possess, and so B in unable to obtain any information about s'.

If f has more than one term, there are two cases:

First, suppose $f = f_0 \vee f_1$. f_0 and f_1 necessarily have fewer terms than f. Because B does not satisfy f, he must satisfy neither f_0 nor f_1, and so by induction learns nothing about s' from either of them. Because the shares of f_0 and f_1 were chosen independently, no joint information is obtained, so B obtains no information about s'.

Otherwise, $f = f_0 \wedge f_1$. f_0 and f_1 necessarily have fewer terms than f. Because B does not satisfy f, he must not satisfy one or more of f_0 or f_1. Let s_0 be the share split by f_0 and s_1 be the share split by f_1. Because s_0 and s_1 are chosen uniformly such that $s_0 \oplus s_1 = s'$, knowledge of s_0 or s_1 alone does not yield any information about s'. The prefix is chosen at random independant of s', so it yields no information about s' either.

By induction on the number of terms in f, a recipient who does not satisfy f learns nothing about s'. □

THEOREM 5.2. *A recipient B who does not satisfy the chosen access structure f is unable to recover the message m.*

PROOF. The recovery of m depends on knowledge of the secret s'. Therefore this theorem clearly follows from lemma 5.1. □

As well as protecting the contents of the message from unqualified recipients, our secret sharing scheme provides the ability to better conceal the content and structure of the policy used to construct a hidden credential ciphertext.

THEOREM 5.3. *When using the secret sharing scheme defined above, our hidden credential system has at least partial policy indistinguishability.*

PROOF. Bluff shares are, by definition, uniformly and independently distributed over $\{0,1\}^l$. Shares resulting from invalid decryptions of V_i are also uniformly and independently distributed over $\{0,1\}^l$ because H_2 was not, with overwhelming probability, called with those arguments in the encryption phase and is not called more than once with those arguments in the decryption phase. So if an adversary does not even partially fulfill a policy $\mathcal{P}$ then all possible shares $s_{ij} = V_i \oplus H_2(\hat{e}(rP, sig_j), i)$ will be distributed uniformly and with no correlation to the policy structure due to the hash function H_2. The probabalistic value rP and the master secret s' are also chosen at random and independant to the policy being used. Thus an adversary's view will be the same for all policies, so he cannot have an advantage in winning a partial indistinguishability game. □

Our system actually has more than just partial policy indistinguishability. We note that valid secret shares, with the exception of the master secret, are uniformly distributed over $\{0,1\}^l$ because both the prefix and the random pad are chosen uniformly over $\{0,1\}^{l_p}$ and $\{0,1\}^{l-l_p}$ respectively. Consequently, if a recipient is able to recover at most one valid share, he is unable to recognize it on its own. In other words, he must fulfill at least two terms of the policy before he discovers that he has partially fulfilled it at all. In fact, depending on their relative position in the policy structure, the recipient may be able to satisfy any number of insufficient terms and still not be able to tell that he has satisfied anything at all. In any case, a recipient discovers at most only those parts of the policy that he fulfills.

6. IMPLEMENTATION

We implemented the secret splitting scheme described in the previous section using Java and the Stanford IBE library (see http://crypto.stanford.edu/ibe/download.html). Our implementation includes a hidden credential aware HTTP server and web proxy. Client HTTP requests are processed by the proxy and a policy is created based on the requested URL. If the proxy is able to decrypt the web server's response, it returns the decrypted response to the client. Otherwise it returns an authentication error and explanatory page. We also created a server-side proxy that can provide hidden credential-based access control to any web server. Our implementation is released under the GPL and is available on our website.

The runtime of the hidden credential algorithms is dominated by the elliptic curve operations used to implement the Boneh/Franklin IBE. In our implementation, 95-98% of the total runtime is spent calculating pairing operations. In the design of the original hidden credential implementation [8], encryption required one elliptic curve multiplication, one

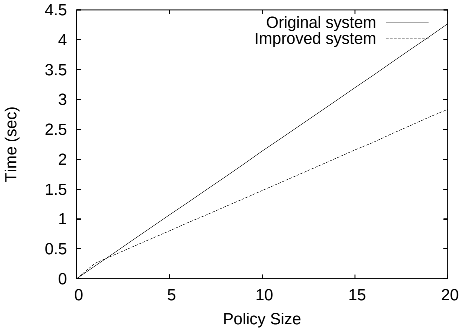

Figure 1: Average encryption time vs. policy size.

elliptic curve pairing, and one modular exponentiation for each term in the policy. Using the optimization discussed in this paper, only one elliptic curve multiplication needs to be computed for the entire policy, and a single pairing and exponentiation are then performed for each unique term within the policy. Decryption sees the larger benefit, however. One elliptic curve multiplication is required for a ciphertext, and one pairing is then used for each credential relevant to the transaction, regardless of the number of secret shares in the ciphertext. The original implementation required one multiplication and pairing for each *combination* of a secret share and relevant credential.

We conducted experiments to measure the actual encryption and decryption performance of hidden credentials on an 867 MHz G4 running OS X 10.3.3 using the security parameters recommended in [6] and in the documentation of the Stanford IBE library (i.e., letting p be a 512 bit prime for the underlying field F_{p^2}, and using a 160-bit subgroup of an elliptic curve over this field). We varied the policy complexity from 1 to 20 unique terms in a policy, with a 50-50 ratio of ANDs and ORs randomly generated for each policy. Sixteen-bit prefixes were used in the secret splitting scheme.

Figure 1 shows the time required for an encryption operation across various policy sizes. For a single policy, there is a small overhead for the optimized approach compared to the original design. As the policy complexity grows, the optimized system performs $n-1$ fewer elliptic curve multiplications than the original system, where n is the number of terms in the policy. This explains the increasing difference in performance between the two systems as the policy size increases.

The impact of the optimized design on decryption is even more significant than the impact on encryption. Figures 2 and 3 show the decryption time for the original hidden credential design and an optimized design, respectively. Previously, unless the policy was overtly given, one had to attempt decryption of each share with each possible credential resulting in up to mn elliptic curve operations for an n-term policy and m candidate credentials. In practice, the number of operations required depends greatly on the policy structure—if the first term of an AND cannot be fulfilled, the rest of the expression is not explored, but every possible combination must be tried for an OR. In the optimized design, a single pairing is performed for each relevant credential, regardless of how many terms are in the policy.

Pre-computation can be used to speed up the Tate pairing when the first argument remains constant across several pairings. Both the old and the new system can take advantage of this fact, but the performance difference remains similar. In the old system $O(n)$ pre-computations are required for the $O(mn)$ (faster) pairings, and in the new system a single pre-computation can then be used for all $O(m)$ pairings.

The decryption experiments varied the number of candidate credentials from 0 to 25. The experimental results in figures 2 and 3 reveal an order of magnitude performance improvement using the new design as the policy size and number of candidate credentials increase.

It should be noted that all the ciphertexts used in figure 3 are indistinguishable from other ciphertexts with the same number of secret shares, while those in figure 2 leak policy structure. Also, had any of the simple policies been reused in the expression, the performance benefits for encryption and decryption using the optimized design would have been even greater.

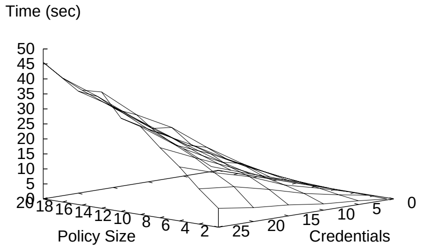

Figure 2: Original system: Decryption time as a function of policy size and the number of credentials held.

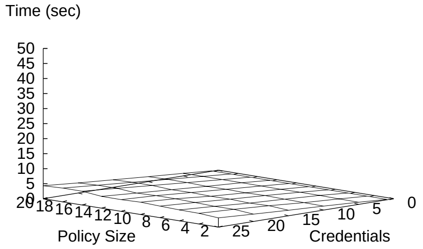

Figure 3: New system: Decryption time as a function of policy size and the number of credentials held.

7. CONCLUSIONS AND FUTURE WORK

We have drastically improved the efficiency of hidden credentials by reusing randomness in a way that does not compromise the security of the system. The number of elliptic curve operations required now depends only on the number of credentials relevant to a transaction and is constant over a change in policy size or complexity.

We also pioneered a perfect monotonic secret splitting scheme where the relevant shares and the corresponding Boolean expression are only revealed as relevant pairs of shares are discovered. Using this secret splitting scheme we are able to extend indistinguishability to complex policies. This secret splitting scheme has applicability outside hidden credentials which we will explore in future work.

Hidden credentials have advantages in policy, credential, and resource protection that no other system we are aware of has. The advances we presented here are essential to achieve the full potential of concealed policies and make the use of hidden credentials a usable model for protecting sensitive resources.

Reducing prefix length in our secret splitting scheme increases anonymity but also increases overhead and the probability of a runaway table. We plan to investigate more fully the exact relationship between prefix length, number of shares, and the probability that the secret splitting algorithm terminates. This may make it possible to continue to guarantee the algorithm's termination in a reasonable amount of time and at the same time provide a higher degree of policy concealment. We are also considering other optimizations, such as varying prefix length based on the height of an operand in the expression tree to provide high concealment at lower levels while reducing ambiguity at higher levels sufficiently to prevent the total number of shares from exploding. It would also be desirable to provide ambiguity in OR expressions. We are considering other secret splitting schemes that support n-ary expressions and may provide full policy indistinguishability.

In relation to this, we plan to more rigorously define and investigate different levels of policy indistinguishability. The makeup and structure of the policy can be seen as plaintext content, suggesting parallels between policy indistinguishability and the several kinds of ciphertext security. There may also be applicable parallels from the domain of anonymous secret splitting.

There are some scenarios where Alice and Bob need multiple rounds of messages to complete their transaction. Each exchange guarantees that all relevant policies are satisfied before any information is revealed, but continuing a transaction may reveal whether or not one was able to understand the preceding message. The ability to bluff when one does not understand a message is available, but the question of when to stop bluffing is an interesting one that was investigated in [8] but needs to be developed further.

We would also like to enable hidden credentials to use more exotic operations on attribute values, such as greater than and less than, without falling back to disclosure of policy and credential content as other systems require.

8. ACKNOWLEDGEMENTS

Thanks to Hilarie Orman, Rich Schroeppel, Roger Dingledine and the anonymous reviewers for their invaluable contributions.

9. REFERENCES

[1] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret handshakes from pairing-based key agreements. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 180–196, Oakland, CA, May 2003.

[2] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Asiacrypt*, volume 1976 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000. Extended abstract.

[3] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In S. Goldwasser, editor, *Advances in Cryptology - CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer, 1990.

[4] E. Bertino, E. Ferrari, and A. Squicciarini. χ-TNL: An XML-based language for trust negotiation. In *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 81–84, Como, Italy, June 2003. IEEE Computer Society Press.

[5] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-7)*, pages 134–143. ACM Press, Nov. 2000.

[6] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *Proceedings of Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[7] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Lecture Notes in Computer Science*, 1666:537–554, 1999.

[8] J. Holt, R. Bradshaw, K. E. Seamons, and H. Orman. Hidden credentials. In *2nd ACM Workshop on Privacy in the Electronic Society*, pages 1–8, Washington, DC, Oct. 2003. ACM Press.

[9] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, pages 182–189, Boston, Massachusette, July 2003. ACM Press.

[10] K. E. Seamons, M. Winslett, and T. Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Network and Distributed System Security Symposium*, pages 109–124, San Diego, CA, Feb. 2001.

[11] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, volume I, pages 88–102, Hilton Head, SC, Jan. 2000. IEEE Press.

[12] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, November/December 2002.

[13] T. Yu and M. Winslett. A Unified Scheme for Resource Protection in Automated Trust Negotiation. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.