Hidden Credentials

Jason E. Holt
Internet Security Research Lab
Brigham Young University
Provo, UT 84602
isrl@lunkwill.org

Robert W. Bradshaw
Internet Security Research Lab
Brigham Young University
Provo, UT 84602
rwb43@email.byu.edu

Kent E. Seamons
Internet Security Research Lab
Brigham Young University
Provo, UT 84602
seamons@cs.byu.edu

Hilarie Orman
Purple Streak, Inc.
Salem, UT 84653
http://www.purplestreak.com
hilarie@purplestreak.com

ABSTRACT

Hidden Credentials are useful in situations where requests for service, credentials, access policies and resources are extremely sensitive. We show how transactions which depend on fulfillment of policies described by monotonic boolean formulae can take place in a single round of messages. We further show how credentials that are never revealed can be used to retrieve sensitive resources.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls, Authentication*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Algorithms, Security, Theory

Keywords

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

1. INTRODUCTION

Hidden Credentials let Alice encrypt a message in such a way that Bob can only decrypt if he has the right credentials. That is, his credentials *are* the decryption key.

Using ideas from identity-based cryptosystems, Alice constructs the public keys for Bob's credentials based solely on credential names, without help from any outside party and regardless of whether those credentials have actually been issued to him.

This has remarkable consequences. First, it can greatly reduce network overhead, especially compared to traditional trust negotiation, which may require multiple rounds of policy and credential exchange before a resource is released.

Second, it solves the "going first" problem in PKI-based authentication systems, where one side of a connection or the other must be the first to reveal a certificate to a potentially malicious stranger. Instead, Alice constructs the public keys which correspond to the types of credentials she requires Bob to possess, and uses them to encrypt her request for service. If Bob successfully decrypts the request using the requisite credentials, he constructs the public keys which Alice must hold to access the service requested, and encrypts his response accordingly.

Third, since Alice can only decrypt Bob's message if she fulfills his policy, Bob never has to actually see Alice's credentials. Consequently, in some cases Alice can obtain resources using credentials she never reveals to anyone. Bob knows that he delivered an encrypted resource to someone named Alice, but never necessarily learns whether she fulfilled his policy and was consequently able to decrypt the resource.

Fourth, Hidden Credentials can be used to protect sensitive policies. When Bob encrypts a message against Hidden Credentials, he need not specify what public keys he used. Alice must then attempt decryption using each of her credentials. Although this can be computationally inefficient, it assures that if she does *not* possess the right credentials, she learns almost nothing about the policy controlling access to the message. That is, she doesn't learn what credentials she would need to obtain in order to decrypt the message.

Finally, Hidden Credentials protocols avoid some known transaction side-channels wherein Alice's and Bob's behavior leak information about their policies and credentials.

2. RELATED WORK

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

WPES'03, October 30, 2003, Washington, DC, USA.

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.

Our work draws much from the paradigm of Trust Negotiation [3, 9, 10]. 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.

Hidden Credentials can make use of identity-based encryption by using the concatenation $nym||attributes$ as the identity, and the credential issuer's public key as the PKG public key. In section 1.1.2 (Delegation) of [4], Boneh and Franklin describe a similar system.

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 credentials from diverse issuers.

Li, Du, and Boneh describe Oblivious Signature-Based Envelopes [5] 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. However, OSBEs require that Alice and Bob agree on the signature Bob needs to have to decrypt Alice's message. 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 credential in question is extremely sensitive.

3. HIDDEN CREDENTIALS IN TRUST NEGOTIATION

The following examples show how Hidden Credentials can be used to facilitate a transaction which would fail using traditional trust negotiation.

A web server provides information on AIDS to both doctors and AIDS patients. In traditional trust negotiation, after Alice has requested access to the site, the server discloses its policy (namely, that clients must possess a Doctor or Patient credential). Alice reveals her credential to the server, at which point the server grants access. If she is concerned about disclosing such a credential, she may send her own policies which the server must fulfill before she sends her Doctor or Patient credential. In all cases, the server sees Alice's credential before granting access, and learns whether she is a doctor or patient.

If Alice is unwilling to reveal her credential to the server, traditional trust negotiation simply fails. Perhaps the server cannot demonstrate a sufficient level of trustworthiness, or Alice fears it may compile a database of AIDS patients. Using Hidden Credentials, Alice can complete her transaction with a single round of messages, without violating the server's policy and without revealing any of her own credentials. She sends a pseudonym along with her request, and the server responds with the requested page encrypted in such a way that she can only decrypt it if she has a doctor or patient credential issued against that pseudonym. That is, the server constructs the public keys corresponding to the secrets Alice has been issued if she is a certified doctor and a certified AIDS patient. The server encrypts the message so that she can decrypt it if she has either of the secrets. That is to say, the server encrypts the message according to the policy "Alice is a doctor OR an AIDS patient." This approach has the further advantage that if the server is ever compromised, the fact that Alice is an AIDS patient remains safe in all past and future transactions.

More exotic scenarios are possible. Consider a case where Alice and Bob are undercover agents, each trying to determine whether the other is an agent from a friendly organization. Their agent credentials are obviously sensitive, but so are the policies protecting those credentials; non-agents would have no reason to maintain a policy for credentials they don't possess. Even their requests, in this case their desire to learn of the other's credential, are sensitive—asking to see an undercover agent credential is certainly a suspicious request.

Traditional trust negotiation fails miserably here. Alice and Bob have to define policies protecting their requests, credential disclosure policies and the credentials themselves in such a way that they can build up trust gradually from nothing. Imagine the conversation:

Alice: "So, do you travel a lot?"
Bob: "I do, in fact. Are you into martial arts?"
Alice: "Sure! Do you like guns?"
etc.

Alice and Bob each must possess a slew of credentials bridging mundane and ultra-sensitive, each one attested by some certificate authority, disclosing information in manageably small increments. If Alice or Bob fails at any stage, all the disclosures up to that point end up being in vain.

This is where Hidden Credentials really shine. Assume Alice is a CIA agent and Bob is an FBI agent. Alice can simply construct an encrypted message:

"Psst! I'm a CIA agent" encrypted with the policy "Bob is a CIA agent or an FBI agent or in the Secret Service."

Likewise, Bob can respond with:

"Great! I'm in the FBI." encrypted with the policy "Alice is in the FBI or the DEA or the Secret Service."

FBI Agent Bob can successfully decrypt Alice's message using his FBI credential because he fulfills Alice's policy of (cia OR fbi OR secret_service). And in this example, since Bob doesn't trust CIA agents, Alice neither understands his response, nor learns that Bob's policy is (fbi OR dea OR secret_service). Here Hidden Credentials grant and deny access to messages according to the senders' relevant policies directly, rather than failing because of an inability to incrementally build up trust from nothing.

4. CREATING AND USING HIDDEN CREDENTIALS

4.1 Definitions

A Hidden Credential system consists of two setup algorithms, an encryption function and a decryption function. All of these functions follow directly from the Identity Based Encryption scheme of Boneh and Franklin [4]. Here we define the functions which set up a credential issuer:

- $CA\_Create()$
  Create a credential authority or issuer.
- $CA\_Issue(nym,\ attribute)$
  Create a credential certifying that the user identified by nym possesses attribute.

The encryption and decryption functions are as follows:

- $CT = HC_E(R,\ nym,\ P)$
  Encrypts a resource R guarded by a policy P with intended recipient identified by nym. Returns ciphertext CT.
- $R = HC_D(CT,\ \{Cred_1\ ...\ Cred_n\})$
  Decrypt CT, returning R iff $\{Cred_1\ ..\ Cred_n\}$ contains credentials issued with respect to nym which are sufficient to satisfy P.

4.2 A Concrete System

In this section we propose a Hidden Credential system using the FullIdent system from [4]. In our system, a certificate authority is simply a Private Key Generator (PKG). A public value $params$ is conventionally agreed upon for use by all PKGs, almost identical to the value specified for FullIdent.

$$params =< q, G_1, G_2, \hat{e}, n, P, H_1, H_2, H_3, H_4 >.$$

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 $G_1 \times G_1$ *to* G_2, P is an arbitrary generator in G_1 and H_1, H_2, H_3, and H_4 are cryptographic hash functions such that $H_1 : \{0,1\}^n \rightarrow G_1^*$, $H_2 : G_2 \rightarrow \{0,1\}^n$, $H_3 : \{0,1\}^n \times \{0,1\}^n \rightarrow \mathbb{Z}_q^*$, and $H_4 : \{0,1\}^n \rightarrow \{0,1\}^n$.

We now define the functions as follows:

- $CA\_Create()$:
  Choose a secret value PKG_{sec}, the private key for the credential issuer. PKG_{sec} is chosen at random from $\mathbb{Z}_q^*$. Publish a public value, $PKG_{pub} = PKG_{sec}P$, the public key for the credential issuer. The issuer may also publish a template, or list of attribute names, for which he issues credentials.
- $CA\_Issue(nym, attribute)$:
  The output of this function, $Cred$, is the secret key given to the credential holder identified by nym certifying the attribute $attribute$. nym is required to be a fixed length string, as a simple way to ensure uniqueness for the concatenation $nym||attribute$.
  $Cred = PKG_{sec}H_1(nym||attribute)$.

Before defining HC_E and HC_D, we first define the functions $HC_{simpleE}(R, nym, P)$ and $HC_{simpleD}(CT, Cred)$, in which the policy P is restricted to a single credential. This will be expanded to more complex policies in section 5.

FullIdent provides encryption and decryption functions which make $HC_{simpleE}$ and HC_{simple_D} trivial to construct.

- $HC_{simpleE}(R, nym, P) =$
  $Encrypt(Params, nym||attribute, R)$
  where $P = \{attribute, PKG_{pub}\}$ and
  $Params = params \bigcup PKG_{pub}$
- $HC_{simpleD}(CT, Cred) = Decrypt(Params, CT, Cred)$
  where $Cred$ is the credential certifying $attribute$ about the recipient identified by nym as attested by PKG.

It is easy to see that $HC_{simpleD}(CT, Cred) = R$ is the decryption of $CT = HC_{simpleE}(R, Recipient, P)$ whenever $Cred$ is the credential issued by $CA\_Issue(Recipient, attribute)$.

4.3 Protecting Sensitive Policies

To protect sensitive policies, $HC_{simpleE}$ must also have the property defined as follows.

Credential Indistinguishability: Let P and P' be elements of the set of possible single-credential policies where $P' \neq P$. Let $CT = HC_{simpleE}(R, nym, P)$ and $CT' = HC_{simpleE}(R, nym, P')$. CT' must be indistinguishable from CT to anyone who does not possess either of the credentials corresponding to P and P'. That is, the recipient of a message learns nothing about the credentials he would need to possess in order to decrypt the message, unless he actually has them.

4.4 Security of $HC_{simpleE}$

Theorem 4.1. *$HC_{simpleE}$ is secure against an adaptive chosen ciphertext attack by one able to obtain an unlimited number of other, arbitrary, credentials.* Specifically, an adversary able to adaptively make an unlimited number of decryption queries and adaptively obtain an unlimited number of arbitrary private keys (credentials) does not have a non-negligible advantage in guessing any information contained in a message, M, encrypted with a key (policy) for which he does not possess the corresponding private key (credential).

Proof. The security of $HC_{simpleE}$ as a cipher follows directly from the security of the *Encrypt* function of the FullIdent system as defined in section 4.2 of [4]. Theorem 4.4 in the same section proves that FullIdent is a chosen ciphertext secure IBE system (IND-ID-CCA), as defined in section 2 of the paper. The proof uses the random oracle model, and assumes the Bilinear Diffie-Hellman problem is hard in the groups G_1 and G_2 over $\hat{e}$. A concrete system for which these properties hold is given in section 5 of [4].

Corollary 4.2. *An adversary able to make decryption queries and obtain an unlimited number of arbitrary credentials does not have a non-negligible advantage in forging a credential he has not obtained.* In other words, Hidden Credentials are unforgeable.

Proof. This follows directly from theorem 4.1, as an adversary with a non-negligible advantage in forging credentials would have a non-negligible advantage in obtaining information about a message encrypted with a policy for which he does not possess the appropriate credential.

The only property that remains to be proved is Credential Indistinguishability.

Theorem 4.3. *Let P_0, P_1 be elements of the set of possible single-credential policies where $P_0 \neq P_1$. Let $CT = HC_{simpleE}(R, nym, P_b)$ where b is chosen at random from $\{0,1\}$. Assuming the random oracle model for all hash functions, if $HC_{simpleE}$ is implemented using the FullIdent system, an adversary does not have a non-negligible advantage in determining whether $CT = HC_{simpleE}(R, nym, P_0)$ or $CT = HC_{simpleE}(R, nym, P_1)$ unless he has the credential corresponding to P_0 or P_1.*

Proof. In the FullIdent system, the ciphertext given by $Encrypt(Params,\ ID,\ M)$ is the tuple

$$\langle U, V, W\rangle = \langle rParams_P, \sigma \oplus H_2(g_{ID}^r), M \oplus H_4(\sigma)\rangle$$

where $g_{ID} = \hat{e}(H_1(ID), PKG_{pub})$, $r = H_3(\sigma, M)$, and σ is chosen at random from $\{0,1\}^n$.

It is easy to see that an adversary able to compute g_{ID}^r given a ciphertext for which he does not possess a valid credential would be able to decrypt the message, contradicting theorem 4.1. Assuming H_2 is a random oracle, this implies that he would not be able to obtain any significant information about $H_2(g_{ID}^r)$ or $\sigma = V \oplus H_2(g_{ID}^r)$. Knowledge of M does not help in learning about σ, as H_4 is a one-way function. Because σ is chosen at random from $\{0,1\}^n$, and the adversary has no information about the value of σ, this value hides all information about $H_2(g_{ID}^r)$. But $H_2(g_{ID}^r)$ is the only part of the ciphertext that depends on the P_b, so an adversary cannot have an advantage on a guess for b greater than his advantage in distinguishing messages for which he does not have the necessary credentials, which has already been proved to be negligible in the preceding theorem.

5. SATISFYING COMPLEX POLICIES

Policies guarding resources are often involve multiple credentials. For example, access to a library might be given only to faculty members or students that are currently research assistants. These kinds of policies can often be expressed as monotonic boolean expressions. Hidden Credentials can handle these kinds of policies.

Given a resource R and a complex policy P governing disclosure of R, we define a function $HC_E(R, nym, P)$ which ensures that the recipient receives R if and only if he fulfills P. This recursive definition allows us to construct policies which are monotonic boolean functions with the addition of {m,n} threshold operations as described by Benaloh and Leichter in [2]. For example, P might be defined as requiring the credentials

$$(C_1 \text{ AND } C_2) \text{ OR } (C_4 \text{ AND } MofN(2, C_5, C_6, C_7)).$$

The function HC_E that is able to handle these complex policies is defined as follows:

5.1 Definition of HC_E

To increase readability, we omit the *nym* parameter in each call to HC_E and $HC_{simpleE}$ in this subsection.

- If P is the simple policy requiring a single credential C, then $HC_E(R, P) = HC_{simpleE}(R, P)$.
- If $P = (P_1 \text{ OR } P_2)$, then $HC_E(R, P) = \{HC_E(R, P_1), HC_E(R, P_2)\}$
- If $P = (P_1 \text{ AND } P_2)$, then $HC_E(R, P) = HC_E(HC_E(R, P_2), P_1)$
- If $P = MofN(m, P_1, ..., P_n)$, then $HC_E(R, P) = MofN(R, m, n, HC_E(s_1, P_1), HC_E(s_2, P_2), ..., HC_E(s_n, P_n))$

  where $MofN$ is the m-of-n threshold encryption of R, and $s_1, ..., s_n$ are the participant shares defined by the threshold scheme.

5.2 Security of HC_E

Theorem 5.1 *HC_E is secure against an adaptive chosen ciphertext attack by one able to obtain an unlimited number of other, arbitrary, credentials.*

Proof. We proved in theorem 4.1 that $HC_{simpleE}$ is secure against an adaptive chosen ciphertext attack by one able to obtain an unlimited number of other, arbitrary, credentials. It is clear that encryption of an already-encrypted value (as in (P_1 AND P_2)) is at least as secure as encrypting under just one of these values. Due to the random values used to make $HC_{simpleE}$ secure against chosen ciphertext attacks, multiple encryptions of the same value (as in (P_1 OR P_2)) are also secure. Likewise, if $MofN$ is secure, then encrypting the participant shares with HC_E is not a problem. Because HC_E is defined recursively as a combination of many HC_{simple}s in a way that does not compromise their integrity, it too is secure against an adaptive chosen ciphertext attack by one able to obtain an unlimited number of other, arbitrary, credentials.

Although the individual credentials needed to satisfy a policy can be hidden by using a $HC_{simpleE}$ with credential indistinguishability, parts of the structure of the boolean expression of the policy may be learned by any recipient. That is, an unqualified recipient may be able to deduce that a message's policy is of the form $((X\ AND\ Y)\ OR\ Z)$ even though he can't decrypt any part of the message. But he won't be able to learn what credentials are needed to fulfill X, Y, or Z unless he possesses them.

5.3 Non-Sensitive Policies

This protection of sensitive policies is a disadvantage to the recipient, since he has to attempt decryption with each of his credentials. But if the sender doesn't mind revealing the policy argument to $HC_{simpleE}(R, nym, P)$, (that is, the sender doesn't mind if the receiver knows that P must be satisfied to decrypt R), she can simply append P to the output of $HC_{simpleE}$. The recipient can then immediately determine what credential to use for decryption.

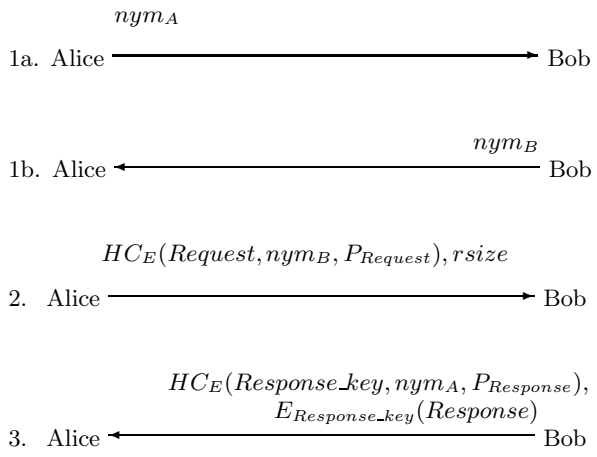
We can implement this optimization simply, by adding a case to the definition of HC_E:

- If P is *not* a sensitive policy, then $HC_{simpleE}(R, nym, P) = \langle HC_{simpleE}(R, nym, P), P\rangle$

6. PROTOCOL

Here we define a simple protocol allowing Alice to request a resource from Bob. Alice's request and policy are protected, as well as Bob's resource and policies.

Other protocols are possible, such as one-way protocols in which Alice sends Bob a message without expecting a response, and multi-round protocols in which Alice and Bob wish to continue communicating after step 3. These protocols have interesting implications which are discussed in Appendix A.



Explanation:

1. First, Alice and Bob exchange nyms. These could be one time use pseudonyms if Alice and Bob are concerned about dossiers. Alice's credentials C_A, and Bob's credentials C_B have been issued with respect to these nyms. This step is unnecessary under some circumstances; for example, credentials could be issued such that the nym corresponds to the IP address of the holder. In such cases, the nym can be obtained from context.

2. Alice sends her request, encrypted according to the policy protecting the request. $rsize$ indicates the length of the resource she wants returned.

3. Bob responds with *Response*, a message, padded if necessary, of length $rsize$. Response is encrypted with *Response_key*, a randomly generated symmetric encryption key. *Response_key* is itself encrypted with HC_E according to $P_{Response}$, the policy governing release of *Response*, which is the policy guarding *Resource* ANDed with all the policies guarding the sensitive credentials he used in decrypting *Request*.

6.1 Preventing a Side Channel Attack

If Bob was unable to recover *Request* (that is, he doesn't possess enough credentials to fulfill $P_{Request}$), he creates a random *Response* of length $rsize$. He sets:

$$P_{Response} = nak$$

where *nak* specifies a credential which is never given to anyone. Thus, Alice will not be able to recover *Response_key* and thus learn that Bob was unable to fulfill $P_{Request}$ and decrypt *Request*. This protects Bob's privacy by preventing Alice from inferring whether Bob possesses credentials in $P_{Request}$ based on his behavior. That is, if Bob always fails to respond when he doesn't understand a request, Alice can tell whether he fulfilled the policy protecting her request based on whether he responds at all.

6.2 Protocol Example

Let $AllCreds = \{C_1 ... C_n\}$, the set of all possible credentials, regardless of issuer. Let C_{iP} be the policy requiring the credential C_i and let P_{Ci} be the specific policy Alice or Bob may have guarding the release of credential C_i (that is, Alice's policy specifying what credentials Bob must have in order to learn that Alice possesses C_i). $C_{iP}*$ indicates that the policy C_{iP} is sensitive; the sender doesn't want the recipient to know that credential C_i is needed to decrypt the message unless the recipient actually has C_i.

Alice has a sensitive resource request, *Request*, requesting resource *Resource*. *Request* is protected by a policy, $P_{Request}$. This policy itself has sensitive terms, which Alice only wishes to reveal to Bob if he fulfills them. Alice also has a set of credentials, $C_A \subset AllCreds$, which she never reveals under any circumstances.

Bob has a sensitive resource, *Resource*, which is also protected by a policy, $P_{Resource}$. Bob has a set of credentials, $C_B \subset AllCreds$, and a policy P_{C5} which determines when a particular credential of his, C_5, may be revealed.

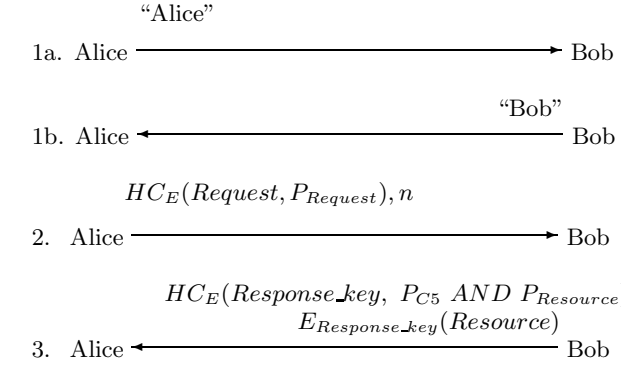
Let $P_{Request} = C_{1P}$ AND ($C_{2P}*$ OR C_{5P})
Meaning: Bob only learns *Request* if he possesses C_1 and either C_2 or C_5. C_2 is sensitive, so don't let Bob know that it's required unless he has it.

Let $P_{Resource} = C_{6P}$ AND C_{9P}
Meaning: Alice only learns *Resource* if she possess C_6 and C_9.

Let $P_{C5} = C_{7P}*$ OR C_{8P}
Meaning: Alice only learns that Bob has credential C_5 if she has C_7 OR C_8. C_{7P} is sensitive.

Let $C_A = \{C_2, C_6, C_7, C_9\}$ and $C_B = \{C_1, C_5, C_7\}$

Protocol:



Explanation:

1. Alice and Bob exchange nyms, as described in the last section. Alice's credentials C_A, and Bob's credentials C_B have been issued with respect to these nyms.

2. Alice sends *Request* to Bob, encrypted with HC_E such that he can only recover *Request* if he fulfills $P_{Request}$.

3. Bob examines Alice's message. HC_E has produced:

$$\langle CT_1, C_{1P} \rangle$$

That is, a ciphertext CT_1 encrypted against credential $C1$. Since $C_1 \in C_B$, Bob uses it to decrypt the CT_1 and recover:

$$\{CT_2, \langle CT_3, C_{5P} \rangle\}$$

That is, 2 ciphertexts encrypting the same plaintext. The first requires a credential which Alice is unwilling to specify. The second requires credential C_5.

Bob has C_5, so he decrypts the second ciphertext to recover Request. Feeling curious, he attempts to decrypt the first ciphertext using his other credential, C_7. He fails, learning that C_7 was obviously *not* the credential required. He never learns, however, that he would have had to have C_2 to have succeeded in decrypting the first branch of the OR statement in $P_{Request}$.

Examining Request, Bob discovers that Alice wants *Resource*. *Resource* is protected by $P_{Resource}$.

He also notes that in responding he will implicitly reveal that he possesses C_5, since otherwise he wouldn't have been able to recover *Request*. Since C_5 is protected by P_{C5}, he combines it with $P_{Resource}$ and sends his response to Alice:

$$HC_E(Response\_key, P_{C5} \text{ AND } P_{Resource})$$
$$E_{Response\_key}(Resource)$$

P_{C5} AND $P_{Resource}$ expands to (C_{7P} OR C_{8P}) AND (C_{6P} AND C_{9P})

Thus, Alice sees:

$$\{CT_4, \langle CT_5, P_{C8} \rangle\}$$

Since $C_8 \notin C_A$, Alice must try to decrypt CT_4 using each of her credentials. C_2 and C_6 fail, but C_7 succeeds, producing:

$$\langle CT_6, P_{C6} \rangle$$

She uses C_6 to decrypt this ciphertext, producing:

$$\langle CT_7, P_{C9} \rangle.$$

Since she also has C_9, she succeeds in obtaining *Response_key* and decrypting *Resource*.

Note that Bob never learns about any of Alice's credentials, even though Alice used 3 of them in recovering the requested resource!

7. IMPLEMENTATION

We implemented a Hidden Credential system in Java, using the Stanford IBE library.

Our system provides a Hidden Credential-aware HTTP server and web proxy. Client HTTP requests are processed by the proxy and a request policy is created based on the 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 will be released under the GPL.

8. PERFORMANCE

The performance of $HC_{simpleE}$ and $HC_{simpleD}$ are essentially the same as *Encrypt* and *Decrypt* in [4]. HC_E and HC_D require as many calls to $HC_{simpleE}$ and $HC_{simpleD}$ as are necessary to satisfy the specified policy. The biggest inefficiency occurs when policies are deemed to be sensitive; in that case the recipient is left to attempt decryption of the ciphertext with each of his credentials. This could be a significant computational cost for users with many credentials.

In our implementation, 90-95% of the processing time is spent executing the IBE functions, despite the fact that the IBE library is written in C and the rest of the system uses Java. On an 867 Mhz G4 running Mac OS X 10.2.6, a single IBE encryption/decryption takes about 200 ms. In a simple case, where both the request and the resource are guarded by a single non-sensitive policy, the time it takes to decrypt the request and then encrypt the response averages about 450 ms. This scales linearly with the number of non-sensitive policies involved. For example, a complex policy with six credentials required for each side takes about 2.5 seconds. If sensitive policies are involved, the performance depends on how many credentials the decrypting party owns. A user may have to try almost all of his credentials for each sensitive credential in the policy. On the same system, this takes an average of 750 ms per undisclosed policy for a user with 6 credentials. Two undisclosed policies take about 1600 ms, and so on.

In terms of network traffic, Hidden Credentials can significantly improve upon traditional trust negotiation. Rather than incrementally revealing policies and credentials over multiple message exchanges, Hidden Credentials require only two message exchanges, regardless of the complexity of the attendant policies; one to exchange nyms, and another to exchange request and response.

9. FUTURE WORK

Hidden Credentials have the unique property that Alice can send Bob a message dependent on certain credentials without ever learning whether he actually has those credentials. This property has implications in many aspects of a credential system. For example, in section 6, we require that Bob respond to Alice's query whether or not he is able to decrypt it. This allows Alice's policy to be respected and Bob to send a response which upholds his policies in a single round of messages, without letting Alice make significant inferences about his credentials unless she satisfies the policies governing their release.

In some situations, Alice and Bob need more than a single round of messages to complete their business. Alice may

be requesting a physical resource or service from Bob, or she may wish to start up an interactive chat session with him after authentication. In either case, Bob must discover whether Alice has received his response to her request. Appendix A informally discusses the ways in which information leaks like the one prevented in section 6 can be avoided for this type of transaction. Future work will expand these techniques and give them more formal treatment.

Credential Indistinguishability also has implications for complex policies. In [2], secrets in AND connectives are split using an XOR with a random pad. This has advantages in terms of formal security over our approach of doubly encrypting, but has the disadvantage that a recipient who fails to fulfill the left operand of the AND operator could still attempt to decrypt the right operand. This doesn't let him recover the resource, but it does reveal more about the sender's policy than our system, since the recipient can't attempt to decrypt the inner encryption without first decrypting the outer. Future work will examine these types of partial policy discovery in more detail.

More exotic policy operations also present an avenue for future work. For example, a GREATER_THAN operator could be constructed for small domains. Alice's "Age" credential could be issued as a set of credentials representing the bits of her actual age; for an age of 19, she would receive "16", "2" and "1" age-bit credentials. Bob's policy of AGE > 18 could then be satisfied by constructing a policy requiring that Alice possess age-bit credentials (64 OR 32 OR (16 AND 8) OR (16 AND 4) or (16 AND 2)). Alice could fulfill Bob's policy without revealing her exact age. Many other kinds of constructions may also be possible.

Complex policies may also be achieved by more efficient methods. For example, it may be possible to implement AND and OR connectives within the $\hat{e}$ function inside Encrypt() in FullIdent.

Credential revocation is also worth considering in a system which protects its users against inferences based on behavior. If Alice is very concerned that others may be able to infer things about her policies, she may need a way to receive updated CRLs without arousing suspicion.

10. CONCLUSION

We have shown how Hidden Credentials have features not found in other credential systems. In particular, Alice can be assured that her complex disclosure policies will be fulfilled without conveying any information to her about Bob's credentials or even his ability to decrypt her message.

We have also presented a protocol whereby Bob can respond to Alice's request for service without revealing whether he was actually able to understand that request, thereby preventing Alice from inferring what credentials he has based on his behavior.

And perhaps most significantly, we have shown how Hidden Credentials can be trivially constructed from any identity-based cryptosystem which satisfies Credential Indistinguishability.

11. ACKNOWLEDGEMENTS

Thanks to Ninghui Li and the anonymous reviewers for feedback on this paper. This research was supported by DARPA through Space and Naval Warfare Systems Center San Diego grant number N66001-01-1-8908.

12. REFERENCES

[1] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, H. C. Wong. *Secret Handshakes from Pairing-Based Key Agreements*. IEEE Symposium on Security and Privacy (Oakland 2003), Oakland, California, June 2003.

[2] J. C. Benaloh and J. Leichter. *Generalized Secret Sharing and Monotone Functions*. Proceedings of Crypto 1988, Advances in Cryptology, Lecture Notes in Computer Science, Vol. 403, S. Goldwasser Ed., Springer-Verlag, pp. 27-35, 1990.

[3] P. A. Bonatti, P. Samarati. *Regulating service access and information release on the Web*. Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens, Greece, November 2000.

[4] D. Boneh and M. Franklin. *Identity based encryption from the Weil pairing*. Extended abstract in proceedings of Crypto 2001, Advances in Cryptology, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 213-229, 2001.

[5] N. Li, W. Du, D. Boneh. *Oblivious Signature-Based Envelope*. ACM Symposium on Principles of Distributed Computing (PODC 2003), Boston, Massachusetts, July 2003.

[6] K. E. Seamons, M. Winslett, and T. Yu. *Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation*. Network and Distributed System Security Symposium, San Diego, CA, February 2001.

[7] W. H. Winsborough and N. Li. *Protecting Sensitive Attributes in Automated Trust Negotiation*. Proceedings of ACM Workshop on Privacy in the Electronic Society, Washington, DC, November 2002.

[8] W. H. Winsborough and N. Li. *Towards Practical Trust Negotiation*. Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), Monterey, California, June 2002.

[9] W. H. Winsborough, K. E. Seamons, and V. E. Jones. *Automated Trust Negotiation*. DARPA Information Survivability Conference and Exposition, Hilton Head, SC, January 2000.

[10] 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, Volume 6, No. 6, November/December 2002.

[11] T. Yu, M. Winslett. *A Unified Scheme for Resource Protection in Automated Trust Negotiation*. IEEE Symposium on Security and Privacy (Oakland 2003), Oakland, California, June 2003.

APPENDIX

A. APPENDIX: PREVENTING INFORMATION LEAKAGE IN COMPLEX TRANSACTIONS

Hidden Credentials can be used in more than just single round transactions. As mentioned, expanding this system to multi-round transactions has some interesting implications. The primary difficulty in this scenario is that the very act

of continuing a dialog may leak information about what was understood, and consequently, what credentials one has.

If a transaction requires more than a single round of communication, it may become difficult to predict just how many rounds the transaction will really need. Transactions that need more than a single round of communication include the following:

- Alice makes a sensitive request for a sensitive *physical* resource, like entry to a building, using a Hidden Credential. Bob responds with a nonce, also using a Hidden Credential, that she can then use to open the door. Bob can see whether she is able to open the door, so that action is considered another "message" in the transaction.
- Alice wants to open an interactive session with Bob. He sends her a session key using a Hidden Credential. Alice and Bob then begin interacting via their terminals.

In these cases, both Alice and Bob must know that the other has understood all the messages sent to them. But because each received message may depend on a particular Hidden Credential, some new policy may be uncovered on the receiver's end which will then need to be satisfied. Consider the following transaction:

Alice: "Give me a dollar, please."
Bob: HC_E("Sure, if you have credential 1. Say the magic word, 'xyzzy', so I know if you got this message.", $Alice, C_{1P}$)
Alice: HC_E("Okay, 'xyzzy'. But you can only know if I have credential 1 if you have credential 2. Say the magic word 'plugh' so I know you got this.", Bob, C_{2P})
Bob: HC_E("Okay, 'plugh'. but you can only know if I have 2 if you have 3. So you say the magic word 'plover'.", $Alice, C_{3P}$)
etc.

Note that Alice *is* qualified to receive Bob's dollar, since the policy for its release only required credential 1. But Bob wasn't necessarily qualified to know that Alice has credential 1. It turned out that he was, but he could only reveal that to Alice if she had 3, because it implicitly revealed his possession of 2. Note that using the techniques described in section 5, C_{1P}, C_{2P} and C_{3P} could be replaced with boolean expressions of credentials.

This sort of situation creates a problem. What happens if Alice or Bob is unable to understand one of the messages? In that case, the transaction necessarily fails, since the receiver is unable to fulfill the sender's policy. But as we've pointed out, simply dropping the connection could leak information about his or her credentials.

There are several solutions to this dilemma. The one we describe is to progressively reveal credentials as in the example above, but with a limit on the number of exchanges which will take place. There are other possibilities, such as Alice sending nonces representing each of her credentials encrypted with their policies to Bob in order to let her find out beforehand which ones Bob is allowed to see.

A.1 Progressive Multi-Round Transactions

Consider what happens in the previous example if Alice is unable to understand the first message from Bob. If she drops the connection, Bob infers that she doesn't have credential X. If she responds, he infers that she does have X, even if he doesn't understand the response.

Alice could try to bluff, as we described in the case of single-round transactions, sending a message dependent upon a nonexistant credential. But now Bob is in a similar predicament–is Alice bluffing, or is he unqualified to decrypt her valid response? To be safe, Bob would have to respond with a bluff of his own.

Since Alice was bluffing, she *knows* that Bob's response is a bluff. But she still can't close the connection! If Bob knows that Alice only closes a connection when she's sure he's bluffing, he knows that her last message to him must have been bogus. Alice can bluff *again*, but now Bob is in her shoes. As long as each knows how the other will handle a given situation, they can work backwards to the original undecrypted message.

Adding randomness doesn't necessarily help either. If Alice chooses a random number of times to bluff before giving up, there's the chance that the number selected will be the lowest number she allows. If Bob knows that bound, he can once again work backwards. Even if she never reaches that lower bound, after repeated transactions Bob could statistically make an educated guess as to whether or not she possesses it.

The solution is to decide *before the transaction begins* how many rounds it will take, at most. If a received message invokes no further policies, Alice or Bob can simply respond with an unprotected reply; there's no need to continue negotiating. But if either party is unable to decrypt a message, he or she starts bluffing. The other will necessarily follow suit, and they continue until the specified number of rounds has elapsed, at which point the transaction can be safely aborted. Because each party has a finite number of credentials, the upper bound can always be determined in advance, since each further message would require at least one new credential to be disclosed. Since this is true at both ends, the bound can be set to the minimum of the number of credentials possessed by each party (plus any additional value they wish to add to their claim, if they're uncomfortable revealing how many credentials they own).