Minimum Information Disclosure with Efficiently Verifiable Credentials

David Bauer
Georgia Institute of Technology
777 Atlantic Drive NW
Atlanta, GA 30332-0250
gte810u@mail.gatech.edu

Douglas M. Blough
Georgia Institute of Technology
KACB, Room 3356
Atlanta, GA 30332-0765
doug.blough@ece.gatech.edu

David Cash
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280
cdc@cc.gatech.edu

ABSTRACT
Public-key based certificates provide a standard way to prove one's identity, as certified by some certificate authority (CA). However, standard certificates provide a binary identification: either the whole identity of the subject is known, or nothing is known. We propose using a Merkle hash tree structure, whereby it is possible for a single certificate to certify many separate claims or attributes, each of which may be proved independently, without revealing the others. Additionally, we demonstrate how trees from multiple sources can be combined together by modifying the tree structure slightly. This allows claims by different authorities, such as an employer or professional organization, to be combined under a single certificate, without the CA needing to know (let alone verify) all of the claims. In addition to describing the hash tree structure and protocols for constructing and verifying our proposed credential, we formally prove that it provides unforgeability and privacy and we present initial performance results demonstrating its efficiency.

Categories and Subject Descriptors
K.6.5 [**Security and Protection**]: Authentication

General Terms
Algorithms, Management, Performance, Design, Security, Verification.

Keywords
Identity management, identity assertion, credential, hash-tree, Merkle tree, PKI, privacy

1. INTRODUCTION
Personal information is increasingly used to establish identity and authorize transactions in the digital world. At the same time, identity theft and fraud, based on unauthorized disclosure and misuse of personal information, are rampant [14], and individuals are increasingly concerned about providing personal information to every digital entity with which they establish a relationship. The research described in this paper is based on several key principles of identity management:

- first and foremost, users should have the maximum control possible over what personal information of theirs is disclosed in any given on-line interaction,
- next, more reliance can be placed on personal information that is verified by trusted third parties than in self-reported information, and
- last, if verified personal information is to be used, mechanisms to prevent that information from being copied and misused by unauthorized parties are essential.

We assume an architecture in which there are identity providers that verify users' personal information and supply *credentials* that the users can give to service providers [17]. Credentials are a common mechanism for verifying personal information in everyday life. Most people carry multiple physical credentials with them, from drivers' licenses to insurance cards to credit cards. A credential describes some set of attributes about the holder. For the obvious example, a driver's license states that the holder is licensed to drive a vehicle in the licensing state. However, due to drivers' licenses being all but universal, they are used as a general credential. As such, driver's licenses often include unnecessary information, such as the holder's date of birth, address, height, organ-donor status, and social-security number. Electronic credentials can be simple, like a user-name and password, or more complex, like a public key infrastructure (PKI) certificate. A PKI certificate is an electronic document that holds an identity and a public key, and that is signed by a certificate authority (called the issuer). The user holds the associated private key to prove that they are the legitimate holder of the certificate. The user-name and password combination is the most widely used scheme, because of its simplicity. However, this scheme also provides no direct information about the user. A user-name must be attached to a previously made account, and some other form of credential must be used to tie an identity to the account. Such accounts are very seldom shared between different domains, leading users to accumulate many different accounts, often with different user-names and passwords. PKI certificates for users are less common, but can solve the problem of needing to keep track of many different user-names and passwords.

Minimum information disclosure in any given interaction is desirable from a user's perspective, and may even be necessary for a given technology to be widely adopted[9]. Clearly, if a user wants to release the minimum amount of personal information on a given interaction, this rules out a single credential approach, in which each user maintains one credential containing all of their personal information and uses that credential for every interaction. As in [9], we say that a user makes a *claim* about herself when she gives information about one or more personal attributes to a digital entity. Due to the wide variety of personal information that

is used in digital interactions, the number of different possible claims is extremely large. The problem to be solved is, therefore, to provide an efficient and reliable mechanism that allows users to assert arbitrarily many (or at least a large number of) verifiable claims over a sequence of interactions with different digital entities.

As an example of minimum information disclosure, consider the problem of verifying that a user is at least 18 years of age. Clearly, verifying the user's date of birth is sufficient but not necessary, and would reveal a very sensitive piece of personal information that could assist an identity thief in masquerading as that user. We refer to the claim that a user "is at least 18 years of age" as a micro-claim, with the (macro-)claim in this case being "the user's date of birth is xx/xx/xxxx". Many different micro-claims can be derived from a single claim, e.g. the user is "at least 18", "at least 21", "at least 35", "at least 65", etc. One possible approach could be to maintain credentials for a relatively small number of claims and use those to dynamically generate micro-claim credentials as needed for a given interaction. However, securely generating a large variety of micro-claims from a given set of claims is an open problem, not to mention how the micro-claims can be verified without revealing the information in the verifiable claims from which they are derived.

Instead, we adopt an approach where a large set of micro-claims is enumerated statically, and updated dynamically as needed. Instead of generating and maintaining a single credential per micro-claim, which is extremely inefficient from the standpoints of storage space, bandwidth, and computation time, we propose a method wherein a single credential can be maintained that allows the user to dynamically specify an arbitrary subset of micro-claims for a given interaction without revealing the other micro-claims. We also include in this credential, and the protocols that use it, mechanisms to make it very difficult for an attacker to make a copy of the credential and use it to masquerade as the user. In addition to allowing the user to update her set of microclaims as certified by one authority, the credential allows information that is verified by different identity providers to be combined in one structure. This allows users to spread out their personal information across different identity providers, thereby lowering their risk when one of their identity providers experiences a security breach. The details of this new credential mechanism and its associated protocols are provided in the remainder of this paper.

Defining the level of privacy needed by users is a subtle and hotly debated issue . One attempt to quantify privacy is the idea of linkability. Two things – events, transactions, credentials, claims, or users – are linkable if they are known to have an underlying connection. It is necessary for some things to be linked; for example, if a claim is not linked to any user or evidence, it is meaningless. But more often the focus is on unlinkability – when things can't be recognized or shown to be connected, even when they are. For example, consider buying an item with cash versus buying an item with a credit card. When using cash, there is usually no record of who bought the item. If several items are bought in separate transactions, it usually cannot be determined that they were bought by the same person. When using a credit card, there is a clear record of who bought the item. Items purchased at different times and even at different places may be identified as having been bought by the same person. Additionally, the real name/identity of that person is known and usually printed right on the receipt. This is a clear invasion of privacy, and yet credit cards are still extremely popular. Credit cards are popular because they are convenient, and this invasion of privacy is accepted because it is necessary to reduce fraud and abuse in the system.

Our goal here is to design systems that preserve the user's privacy as much as possible, while remaining auditable from the point of the of the service providers. That is, a user should have control over which claims are presented, but repeated use of the user's credential should be linkable. This concession to social practicality gives a strong engineering advantage compared to the other credential systems discussed in the related work section: we are trying to solve a somewhat easier problem. As such, we can focus on making the system faster and more efficient. While complete anonymity, e.g. in the form of unlinkable transactions, is a noble goal, it is our belief that many types of service providers will not accept it as a practical solution. Our approach, by contrast, tries to improve on the current practice by allowing users to minimize the amount of personal information they entrust to service providers while recognizing that release of some personal information may be inevitable.

2. BACKGROUND

2.1 Expected Scenario

We consider a scenario with three types of parties: identity providers, service providers, and users. An identity provider is an entity in a position to make authoritative statements about a user. An identity provider can be a third party certificate authority, a government office, an employer, a professional organization, etc. A service provider is an entity that wants or needs to check the identity of users. A service provider can be any organization, government office, or individual with an on-line presence. The user is the holder of the credential, and is always an individual person in our scenario.

We acknowledge that it is notoriously difficult to deploy a usable and secure PKI to bind keys to users, but it seems necessary without an online third party to attest to a user's claims. The functionality provided by verified claims will hopefully serve as intuitive and strong motivation for users.

2.2 Requirements

The design of our electronic credential was driven by several requirements.

1. The credential must not simply hold a single identifier. Identity is a complex matter, and often a name or serial number is not what is important in one's identity.

2. The user of the credential -- the one whose identity is being proven -- must have as much control over the process as possible. Considering the first requirement, the user should be able to select which attributes of their identity are released to a particular entity.

3. The credential must have some form of copy protection. Using the credential should not place it at risk of being copied.

4. Neither the user nor the service provider should have to contact the identity provider to verify the credential.

5. The credential must be memory and computationally efficient to store and use.

2.3 Related Work

Digital credentials have been proposed before in various forms. More in depth comparisons between our system and these systems can be found later, in Section 3.4.

David Chaum is credited with proposing the first digital credential system[11]. Chaum's system is a pseudonym system, whereby a user can present a different pseudonym to each service provider. The pseudonyms are linked cryptographically such that even colluding service providers cannot link together two pseudonyms belonging to a user, but the user can use a credential assigned to one pseudonym with another pseudonym. Chaum's original proposal did not describe how to implement such a system. A method for implementing the system was later described by Chaum and Evertse based on the RSA crypto-system and a semi-trusted third party [12].

Stefan Brands developed a different form of digital credential, in which a user has a single public credential, but that credential is pseudo-anonymous, even to the issuer [4]. The credential holds attributes that the user can selectively prove to a service provider. Repeated showings of the same credential are linkable, both if shown to the same or different service providers. However, since the credential is issued blind by the identity provider, the effect is that a user has one global pseudonym. The credential can be reissued easily, allowing the user to change the global pseudonym, as permitted by the identity provider. Brands emphasizes that showing a credential is done by zero-knowledge proof, but that has no impact on the comparisons in this paper. Credentica's U-Prove Software Development Kit is based on Brands' work [3].

Camenisch, et al., have proposed and implemented yet another form of digital credential, or more precisely, yet more forms [6-8]. While they describe a system for implementing a Chaum-like pseudonym system, their system is much more flexible, and can be used without pseudonyms. While having significantly better anonymity properties, the algorithms are also significantly slower than Brands' credentials. IBM's idemix system is based on Camenish, et al.'s work [6].

While their aim is very different, the closest design to our credential system is the redactable signature scheme described by Johnson et al in their paper on homomorphic signature schemes [15]. A comparison between our credential and their signature scheme is given later in Section 3.4.

3. DESIGN

3.1 Merkle Hash Trees

The proposed credential is based on Merkle hash trees [16] and standard public-key infrastructure (PKI) certificates. A Merkle hash tree is essentially a binary tree where each internal node holds the hash of the concatenated values of its two children nodes. The leaf nodes hold the data of interest. In this way, a large number of separate data can be tied to a single hash value. In addition, by storing the internal node values (or a subset thereof), it is possible to verify that any of the leaf nodes is part of the tree without revealing any of the other data. Ralph Merkle first introduced this structure as a way to efficiently handle a large number of Lamport one-time signatures. It has since been adapted for uses such as the large-scale time-stamping of documents [2] and tracking data in peer-to-peer networks [10].

A basic Merkle tree is shown in Figure 1. Figure 2 shows the same tree with the nodes and claims labeled for easy reference. Consider verifying claim A in the tree. Starting at the claim and going up the tree, node 4 contains the hash of the claim. Node 3 contains the hash of the concatenation of nodes 4 and 5. Node 2 contains the hash of the concatenation of nodes 3 and 6. And, finally, node 1, the root of the tree, contains the hash of the concatenation of nodes 2 and 9. Therefore, verifying claim A requires the values of nodes 5, 6, and 9, as shown in Equation 1, below. A similar verification path can be made for any of the claims. For example, Equation 2 shows the path for verifying claim F, which requires the values of nodes 2, 11, and 13. As per the nature of the binary tree, the number of nodes and hashes needed for a verification scales with the log of the number of claims.

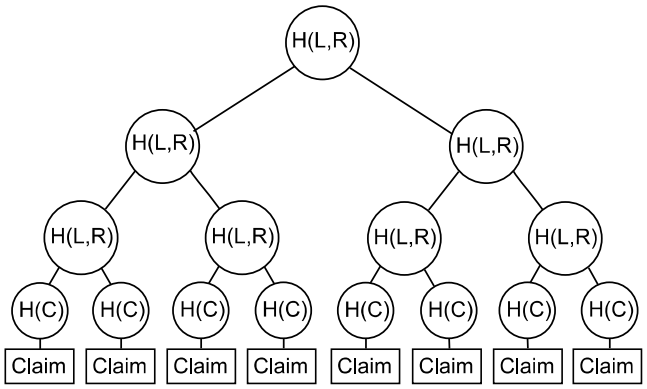

Figure 1. Merkle hash tree with leaf nodes holding hashes of claims.

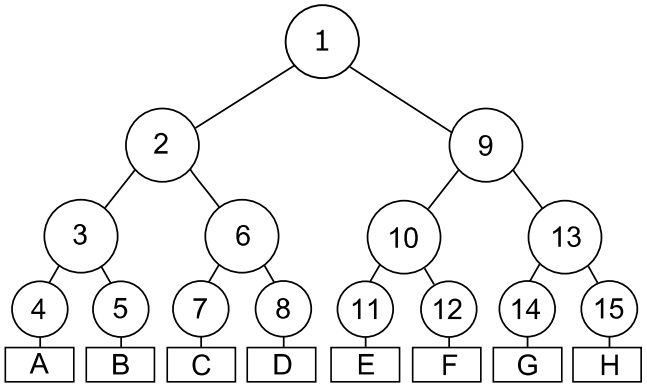

Figure 2. Merkle hash tree with labeled nodes.

3.2 Credential Overview

The credential consists of two parts: a public part and a private part. The public part of the credential is a certificate. The certificate holds information about the issuer, the certification chain for the issuer, the type of certificate, the date range over which the certificate is valid, the user's public key, and a signature of the root node of a Merkle hash tree. The certificate should not

in general hold any data about the user directly, even data as common as a name. As per standard operation, the certificate will be signed by some certificate authority, which is an identity provider in this system. The private part of the credential consists of a private key and a Merkle hash tree whereby all of the leaf nodes are attributes of, or micro-claims about, the identity of the user, who is the credential holder. The Merkle tree structure allows the credential holder to prove any subset of the claims in the tree, with only the single signature on the certificate.

As an additional improvement, a slight modification to the structure will allow the use of a single credential containing claims from a variety of identity providers, without requiring one identity provider to verify all of the claims. Consider again the Merkle tree of claims, but with subtrees coming from different identity providers. For example, one subtree could contain claims certified by a government registry, while another subtree could contain claims certified by an employer. In the basic structure, the identity provider must either see all of the claims or trust the providers of all of the subtrees that they only contain claims relevant to their topic. Neither solution is ideal. To provide a third option, consider adding an optional third branch to some internal nodes of the full tree. (These nodes correspond to root nodes of the subtrees.) The third branch contains a certificate for all of the claims in the subtree rooted in the parent node. When verifying a claim which is in such a subtree, the form is different -- three hashes are concatenated in a node instead of two -- so the verifier knows that this claim is under a different certificate than the credential as a whole. For example, in Figure 3 the left hand subtree contains claims certified by the overall identity provider. The overall identity provider is referred to as the certificate authority (CA). The right hand subtree contains claims certified by some other party, which do not have to be verified by the top level CA. From here on, the term "subtree" is used to refer to a branch containing nodes from a different identity provider, rooted in one of these sub-roots.

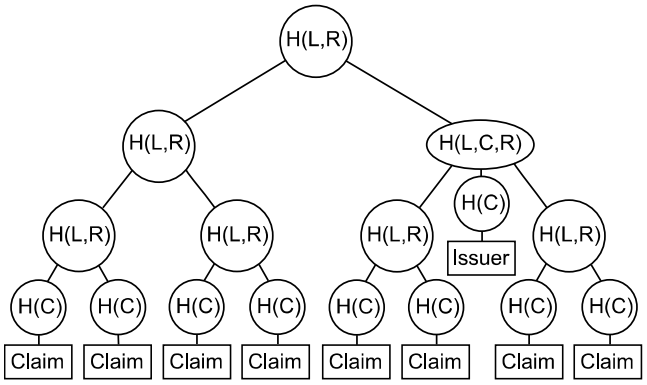

Figure 3. Modified Merkle hash tree with subtree.

We would like to identify a node as being a leaf node without seeing the associated claim. The reason for this is described. This discrimination is achieved by simply appending a bit to the end of the hash, either a one if the node is an inner node, or a zero if the node is a leaf node. For efficiency, our implementation simply overwrites the least significant bit of the hash value, instead of appending another bit. This is equivalent to truncating the output of the hash function used by a single bit (with a negligible decrease in security), and then appending the indicator bit. In order to keep the system secure while still only needing to check the signature on a single certificate, nested subtrees are forbidden. If a service provider encounters a claim in a nested subtree, it should reject the credential.

A tree might have no claims outside of subtrees. In this case, the top level CA is signing that it has verified that the form of tree is correct, that all certificates for subtrees are valid, and that the user possess the private key(s) matching the subtree certificates and the top level certificate. The CA could be fully automated in this situation, allowing users to easily update their credentials when subtrees are added or modified.

3.3 Protocols for the Credential

Creating a credential is both conceptually and computationally easy. In the case where there is a single identity provider for the credential, there are roughly four steps required:

1. Agree on a list of claims.
2. Generate the hash tree for the claims.
3. Verify that the user possesses the private key.
4. Produce and sign the public certificate.

First, the user and identity provider agree on a list of claims. The logistics involved in an identity provider verifying the user claims may be rather complicated, and are beyond the scope of this paper. Second, either the user or the identity provider generates the hash tree for the claims. Random padding must be added to the claims before they are hashed, as discussed in more detail later. With a single identity provider, the tree will always be balanced, bounding the number of interior nodes to the number of claims. The number of hashes needed to generate the tree is therefore bounded to twice the number of claims -- generally a negligible amount of computation time. Third, the identity provider must verify that the user holds the private key matching the public key of the credential. The user can reuse an already generated key pair or generate a new key pair for the credential. The identity provider does not need to ever know the private key. Finally, the identity provider creates and signs the public certificate for the credential.

Random padding must be added to the claims before they are hashed in order to prevent dictionary attacks against parts of the tree. The random padding can be generated and stored in a number of different ways. For example, Johnson, et al. [15], describe a method credited to Goldreich, Goldwasser, and Micali [13] that uses a pseudo-random function whose output is twice the length of its input. A single seed value at the root node is expanded as per the tree branching structure using the pseudo-random function. Each node therefore has a pseudo-random value associated with it that can be used to generate the pseudo-random values for all of its children nodes. A computationally easier method is to use a seed value and the claim index with a pseudo-random function to generate the padding for each claim. Neither of these methods will work well for our system, unfortunately, due to the inclusion of different subtrees within a larger tree. In the most general case, simply storing the random padding for each claim is probably the easiest solution. The overhead is small, just 10-16 bytes per claim. Alternative approaches are possible, but all require the user to have a global secret, keep the private keys

for all certificates, or store extra private information with each credential.

Creating a credential with claims from multiple identity providers is done by integrating credentials as subtrees. One identity provider will be the final one, referred to as the certificate authority (CA). The user creates credentials with all of the identity providers, except for the final one, by the procedure described above. Then, the user creates a credential with the CA. For this final credential, the hash trees from the other credentials are incorporated into the final hash tree. The root nodes for the incorporated subtrees will have a third branch added, each containing the certificate from the original identity provider of that subtree, as shown in Figure 3. (The root hash values in the subtree certificates will no longer match, but that doesn't matter.) All subtrees in the credentials being added must be removed from their trees and added separately to the top level tree. This prevents nested subtrees. The CA verifies the structure of the tree, including the top two leaf nodes of each subtree (as they are needed to calculate the sub-root's hash), as well as the public part of the sub-credentials, and the associated public/private key pairs. In general, the public/private key pairs may be the same as or different from each other and the top level key pair. All that matters is that the user holds the corresponding private keys at the time that the credential is assembled. In a more restricted setting, the public/private key pair may be required to be the same for all subtrees. The CA does not need to see the claims from subtrees.

To provide an example of combining credentials, imagine a user, Alice, who already has a credential containing claims from four different identity providers. Now, Alice is then issued a credential from her employer. The two credentials are shown in Figure 4. Alice's employee credential already has a number of subtrees, containing separate credentials from the engineering department, human resources department, and her own lab. The credential also has a top level claim of her employee ID. Alice goes to a certificate authority to create a new credential from these two credentials. She submits no claims to be verified by the CA and eight sub-credentials to be included in the final credential – four of these are from her previous credential, three were embedded in her employee credential, and the last is the credential claiming her employee ID. The CA will verify the eight certificates, verify that Alice possesses the appropriate private key(s), generate a new tree, and provide Alice with the new certificate.

The protocol for using the credential follows conventional PKI certificate usage. The user connects to a service provider, either over a wide or local area network, and requests some service, sending the public part of the credential. The service provider requests the appropriate identity attributes from the user. The user provides the claims that match the requested attributes and the intermediate node values and path information necessary for the service provider to verify each of the claims. If any of the claims are in a subtree certified by a different identity provider, the accompanying certificates must be included with the set of claims. The service provider verifies that the claims are in the hash tree specified (via the root hash) in the certificate part of the credential. The service provider also verifies the signature on the certificate part of the credential. In order to verify that the user is the holder of the credential, the service provider also verifies that the user possesses the private key which matches the public key claimed by the credential. This can be done by standard methods, such as challenge/response or as part of a secret key agreement operation, as long as the key verification is tied to the specific claims being asserted by the user. This can be done, for example, by hashing all of the claims asserted by the user as part of the challenge field. In addition to the cryptographic verifications, the service provider must of course confirm that it trusts the identity providers to assert the claims in the credential. For example, a claim of an individual's address asserted by the Bar Association would be out of place. Similarly, an assertion by the Department of Motor Vehicles that an individual was a licensed lawyer should not be trusted.

The procedures for creating and using credentials given herein are purposely generic. We prefer to constrain the structure and properties of the credential and leave some flexibility in the procedures for generating and verifying it. As a proof of concept, we have developed specific implementations of these procedures, and initial performance results based on these implementations are reported in Section 5.

3.4 Comparisons to Related Systems

Johnson, et al., define and construct *redactable signatures* using a Merkle hash tree to allow the signature verification of a message even when parts of the message have been deleted [15]. In their example, a body of text is signed such that it can be redacted at some level of granularity (sentence, word, or character making the most sense). They do not appear to have considered using the construct in the context of a credential. While both systems share a core idea – using a Merkle hash tree to hide signed elements – the two systems have significant differences. Both systems use slightly different hash functions (constructs) for the leaf and inner nodes. In their signature scheme, the input to the hash function is specified to match a certain form, which is important to their proof of security. In our credential system, the output of the hash function is specified to match a certain form, which is sufficient meet the requirements of the security proof, while provided the property of being able to identity a leaf node from its hash value

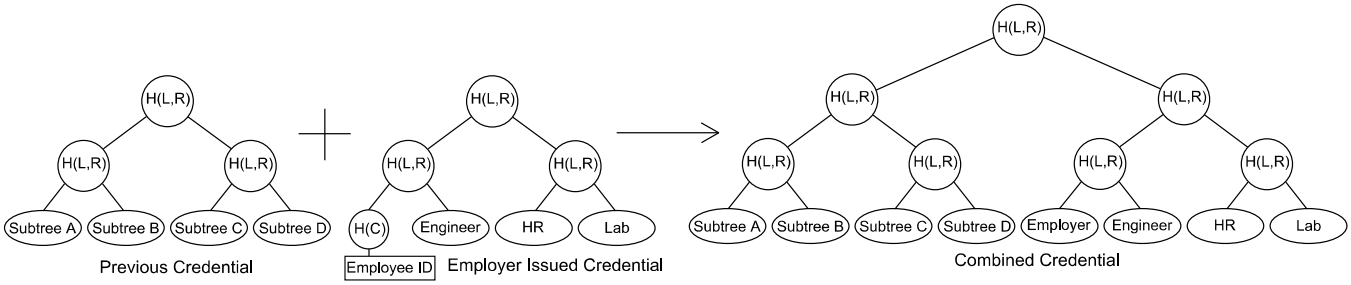

Figure 4. Combining a generic credential with an employee credential.

alone. Subtrees of the type used in the credential system are not used at all in the signature scheme. To the best of our knowledge, the idea of combining trees by modifying the tree structure as we do is novel, and introduces extra complications. The three-child inner nodes (shown in Figure 3) and modified hashing scheme (discussed in Section 3.2) both result from the idea of combining subtrees from different identity providers. Combining subtrees allows claims from many different sources to be in a single credential, under a single certificate.

Brands mentions in his book the idea of using Merkle hash trees to store claims, but dismisses it because it does not provide the properties desired in his system [4].

3.5 Copy Resistance

The credential is strongly copy resistant under normal operations, due to the incorporated public/private key pair. Whenever the credential is used, the private key is used to prove that the user is the authorized credential holder. Since the private key never leaves the user's machine, it can't be copied by the service provider or any third party listening to the exchange. (The private key can of course be copied if the user's machine is compromised, but that is beyond the scope of this paper.)

This copy resistance property extends to preventing standard man-in-the-middle attacks. For example, consider how a phishing site handles conventional one-time passwords or other two-factor authentication methods. A user logs in to what they believe is the legitimate site of their bank, broker, or other service provider, but which is really a phishing site. The user enters their user-name and password and then, either in the same step or in a second step, enters a one-time password from a sheet of paper or electronic device. The phishing site simply passes all the information from the user on to the site being spoofed, and returns the responses from the legitimate site to the user. After the user has successfully logged into the legitimate site, through the phishing site, then the phishing site is logged in as the user and can perform whatever actions it wants.

The standard protection against this type of man-in-the-middle attack is the use of server-side certificates with SSL/TLS. When the user initiates a secure connection to a service provider, the service provider replies with a PKI certificate. The user's client (web browser) usually handles checking the validity of the certificate automatically, interrupting the user only when a problem is detected and simply showing a non-intrusive indicator when the secure connection is setup without problems. Phishing sites have used a variety of techniques to get around server-side certificates, including simply not using SSL/TLS, making the browser look like it is using SSL/TLS when it is not, and obtaining valid certificates for different domains, which can be mistaken for the legitimate domain by an unwary user. All of these tricks are possible because of the disconnect between the authentication system (SSL/TLS), and the thing being authenticated (the service provider). The certificates authenticate the domains used, instead of the actual service providers.

In contrast to the server-side certificates, our credentialing scheme intimately ties the authentication system (the private/public key pair and the certificate as a whole) to what is being authenticated (the credential as a whole). Consider again the phishing site, but with the credential in place of the user-name, password, and one-time password. When the user connects to the phishing site, the phishing site can relay requests to the legitimate service provider and replies back to the user as before. However, when the service provider sets up a secure connection, it will use the user's certificate to setup the session. The phishing site can substitute its own certificate, but then it is no longer impersonating the user. Alternatively, the phishing site can pass on the user's certificate, but then the phishing site loses control of the session. The service provider can setup an authenticated tunnel to the user, so that the phishing site can no longer modify the traffic without being detected.

4. SECURITY

In this section we discuss some potential attacks against implementations and then provide a formal analysis of the underlying cryptographic construction.

4.1 General Security Discussion

The basic attacks that the credential must resist are forgery, theft, privacy compromise, man-in-the-middle, and collusion.

Forgery – a malicious user should not be able to forge a valid credential containing invalid claims.

Theft – the credential should be resistant to theft under standard usage.

Privacy compromise – an attacker should not be able to learn more about the user of a credential than the user chooses to reveal.

Man-in-the-middle – a user should not have to worry excessively about hostile service providers.

Collusion – two or more users working together should not be able to make a claim that no single user in the group can make alone.

During the design process, several specific attacks against earlier versions of our system were discovered. They include dictionary attacks against neighbor nodes, combined replay/man-in-the-middle, hidden subtrees, and the broken hash attack.

Dictionary attacks against neighbor nodes come from the fact that when a credential is used, the hash values of unreleased claims must be provided. Assuming a secure hash function is used, an attacker cannot determine the value of the unreleased claim via cryptanalytic attack. However, since most claims are likely to be in a standard form, a dictionary attack should often be successful. We protect against this type of attack by padding the claims with random or pseudo-random data.

A combined replay/man-in-the-middle attack may be possible when the verification that a user holds the private key for a credential is completely unrelated to the showing of the tree for the credential. In this situation, an attack can perform a replay attack against the showing of the tree, and then a man-in-the-middle attack against the user proving possession of the private key. While this attack either does not apply or would be meaningless to many uses of the credential, it is still a possible attack. We protect against this type of attack by requiring that the verification of the private key be linked to the specific claims being shown.

To maximize the efficiency of showing a credential, we would like to minimize the number of slow, public-key operations performed. The best that we can do is to have only a single certificate verification and a single public/private key verification, regardless of the number of claims and how many different subtrees the claims may be in. As described previously, an identity provider will check the certificates for all subtrees within the tree. Therefore, when the credential is shown, the service provider does not need to recheck those certificates, as long as it trusts the higher level CA. However, consider the case of a nested subtree. An identity provider shouldn't be able to see the claims in subtrees, due to privacy requirements. But this means that a subtree can hide another subtree. In particular, a subtree issued by a bad (but untrusted) identity provider could hide a subtree apparently issued by a trusted identity provider, but in actuality was forged and has a bad signature. This could be prevented by several different protocol changes. The most efficient of these is by preventing the inclusion of a hidden subtree in the first place. Our current recommendation is simply to ban nested subtrees altogether, and have the service providers ensure that credentials which have nested subtrees are rejected.

An alternative solution to the hidden subtree problem allows nested subtrees, but prevents them from being hidden. As described previously, the final bit of the hash is set or cleared to indicate an inner node or leaf node, respectively. When a user wishes for a subtree to be present in a credential, the entire tree down to the leaf nodes must be presented to the identity provider/CA. The claims of the subtrees remain private. As the node type is evident from its hash value, subtrees cannot be hidden during this process. Service providers must make sure this standard is followed by verifying that no inner nodes have hash values appropriate to leaf nodes. In our prototype, this is trivially accomplished by having different methods called to compute the hash value, depending on the node type. Compared to our recommended method, this solution to the hidden subtree problem (trivially) increases the computation and communications cost of issuing a credential.

Another structural attack that was encountered is implementation specific. In our prototype implementation, plain X.509 certificates are used. X.509 certificates are designed to match a directory entry for a particular subject, and their structure is fairly rigid around that purpose. Therefore, our implementation didn't specify any extra details about the hash tree – such as the hash algorithm used – in the certificate itself, but instead stored that information with the tree. This can lead to an attack if a hash algorithm trusted for use in creating the tree is broken. Given the right conditions, an attack could create a tree using the broken algorithm, which could then collide against the root hash stored in a valid certificate, even if the valid certificate was for a tree using a different (and hopefully more secure) hash algorithm. We consider this attack to be fairly minor, because it requires a primitive of the system to be broken badly – in order to find a collision from one hash function to another should effectively require a preimage attack, and not a simple collision. As version 3 of X.509 certificates allows arbitrary extensions, the hash algorithm can be specified in the certificate, anyway.

4.2 Formal Analysis of Security

We provide a rigorous analysis of our credential system. We first formally define the new notions of *unforgeability and privacy for a credential system* and show that our system achieves these under standard cryptographic assumptions.

Unforgeability states that a user cannot convince a service provider that a set of claims is true unless those claims were actually approved by an identity provider. We formalize this in a strong way: we allow adversaries to adaptively obtain an unbounded polynomial number of credentials on sets of claims, and then require that the adversary can not generate a credential on a set that was not contained in one of the valid credentials. Furthermore, the adversary may request that credential trees be combined. The adversary is also considered to have won if it can fool a service provider into accepting a valid claim from one identity provider as a claim from another identity provider which did not approve the claim. For simplicity, we analyze the cases where the claim tree contains no subtrees or is composed entirely of subtrees. The analysis can easily be extended to the case where claims are mixed with subtrees.

Privacy states that no partial information about unrevealed claims is leaked. We will allow adversaries to adaptively obtain credentials and then submit two sets of "challenge" claims of equal size. One of the challenge claim sets is selected at random and then a credential is generated on that set and returned to the adversary. The adversary can ask that parts of the challenge credential be revealed before outputting a guess. Of course, we must restrict the adversary to revealing only claims on which the sets match (otherwise distinguishing is trivial).

We leverage two properties in our analysis that were not considered in the redactable signature scheme of Johnson, et al., which used a similar technique of constructing a Merkle tree and then signing the root [15].

First, we consider the privacy of unrevealed claims, a notion that did not apply to the context of redactable signatures. If we translate Johnson, et al.'s terminology to our context, it was assumed that the adversary *knew* all of the claims in a given tree, and after the user revealed some claims, the adversary would attempt to convince an identity provider that some of the remaining unrevealed claims were in the tree. The security theorem given by Johnson, et al., stated that the adversary could not succeed, despite knowing all of the claims. For unforgeability, we need only a weaker version of this security that prevents an adversary from inserting new claims, and for privacy we must defend against adversaries who do not know all of the user's claims and wish to learn something about the unrevealed claims.

Second, combining trees did not apply in their context and was not considered. For this work, combining trees allows us to save the computation involved in verifying many signatures if the key used to sign the root node is trusted.

In the following definitions, we treat a credential system as a pair of algorithms (KeyGen, SignCred). KeyGen takes as input a security parameter and outputs a unique ID for the identity provider and a public/secret key pair to be used in generating and verifying credentials. SignCred takes as input an identity provider's secret key and a set of claims, and outputs a credential

on the set of claims. Each claim is either an arbitrary string s or a tuple (id, s, σ), where id is an identity provider's unique ID, s is an arbitrary string, and σ is a signature. The second type of claim corresponds to a request to combine a subtree with root s from the identity provider with unique ID id, and σ is the signature on s.

For the following definition we define to two oracles, CreateIDP and RetrieveCred. CreateIDP runs KeyGen and returns the id and public key, and RetrieveCred takes as input a claim set and an identity provider's unique ID and runs SignCred with the corresponding secret key, and then returns the credential.

Definition 1. (**Unforgeability**) Let (KeyGen, SignCred) be a credential system. Then a probabilistic, poly-time adversary $A^{CreateIDP,\mathrm{Re}trieveCred}$ is said to forge a credential if it outputs a tuple $(pk^*, \sigma^*, \{c^*_i\}, \{n^*_i\})$, where pk^* is a public key output by GenIDP, σ^* is a valid signature under pk^*, $\{c^*_i\}$ is a set of claims not signed by their respective identity providers, and $\{n^*_i\}$ is a valid set of intermediate nodes for revealing the claims in a hash tree corresponding to σ^*.

Definition 2. **(Privacy)** Let (KeyGen, SignCred) be a credential system, let Chal be an oracle that when given two claim sets and a identity provider's ID, chooses one of the sets at random and generates a credential on that set, and let Reveal be an oracle that takes as input a claim set and reveals those claims (Reveal may only be called after Chal has been called). Then an adversary $A^{CreateIDP,KeyGen,Chal,Reveal}$ is said to violate the privacy of the credential system if it guesses the set chosen by Chal with probability non-negligibly greater than ½.

THEOREM 1. *If H is a collision resistant hash function and the underlying signature scheme is existentially unforgeable, then the scheme described above is unforgeable except with negligible probability.*

PROOF SKETCH. Above we assumed that a tree is valid only if all of its claims are contained in subtrees or if none of its claims are in subtrees. We will deal with these two cases separately.

*Case 1: **A** outputs no subtree claims.* Here σ^* must be a signature output by the underlying signature scheme under a key output by CreateIDP, because otherwise **A** could be used to break the underlying signature scheme. Moreover, the nodes revealed by **A** must be a subtree of the tree from that query. Otherwise **A** has found a collision in the hash function: either a tree node's input was changed, or if an internal node was used as a leaf, then the appended bit is different, forcing a collision.

*Case 2: **A** outputs a tree containing subtrees.* Here again the signature at the root provided by **A** must have been output by SignCred. **A** must reveal the intermediate nodes leading up to the root, and since every claim is contained in a subtree, **A** must reveal a preimage of the trinary node (the root of the subtree). By the collision resistance of H, **A** must reveal the same public key as part of the preimage of the trinary node. By the same reasoning, the revealed tree must be contained in the honestly generated tree, so the claims must have been signed before and the matching trinary node preimage guarantees that the claim is under the original identity provider. This completes the proof sketch.

We note that the random padding is not needed for unforgeability, but it is necessary for privacy, as discussed below.

Next we argue that the scheme does not leak information about unrevealed claims. Below we will need that the hash function is a *pseudorandom function* when the random padding is viewed as a key.

THEOREM 2. *If $H_k(x) = H(k,x)$ is a pseudorandom function (keyed with the padding k), then the scheme described above is private.*

PROOF SKETCH. Consider an adversary **A** attempting to violate the privacy of the credential system, as defined above. A will obtain credentials on sets until it outputs two challenge sets, which define a symmetric difference of claims which it cannot ask to be revealed.

We define a sequence of "hybrid" versions of the game played by **A**, where in each hybrid we hash one more claim in the symmetric difference with an independent random function instead of with H_k. Since H_k is a pseudorandom function that is only queried once for each randomly chosen k, **A** cannot detect the difference between adjacent hybrids. Then **A** cannot detect the difference between the extreme hybrids, where either all claims in the symmetric difference are hashed with H_k or all are independent random values. But in the latter case, the distribution seen by **A** is the same no matter which of the two challenge claim sets is chosen, meaning that **A** has advantage 0 in that hybrid. Thus **A** has negligible advantage in distinguishing the two claim sets, completing the proof sketch.

5. PERFORMANCE

As per its conception as a practical system, performance has always been a consideration in the design of our credential. Our design focuses on conventional operations (one-way hashes) and minimizes the number of asymmetric/public key operations. In general benchmarking of comparable hash functions (the SHA family) versus public key algorithms (RSA and DSA), done using both OpenSSL and the default Sun Java cryptography provider, the hash functions were more than two orders of magnitude faster than the public key algorithms. This allows our system to be much faster than the digital credential schemes by Brands and Camenisch, et al., which require many more of these expensive public key operations.

The actual implementation of the credential system is of course more complicated than the primitive operations alone. Based on our proof of concept implementation, Table 1 shows timing for the whole operations of verifying a hash tree versus verifying a public key certificate (not including verifying the user's private key). The hash tree used contains 2048 claims. In the first case, all but a single claim is masked, leading to both a very fast execution time (a little over a millisecond) and a very small file size on disk (just a few hundred bytes). In the second case, all 2048 claims are present, requiring 10 times as long to verify. (The time difference is not greater, because the checking algorithm is very general, performing the same setup for checking a full tree as a tree with only a single claim). The tree file for the full 2048

claims is about 31 kilobytes, but most of that (25 kB) is taken up by the claims themselves plus a 32 bit length field for each claim. The tree sizes do not include separate random padding for each claim.

To see the performance advantage of our method, consider a system where a separate certificate is provided for each claim. In the situation of checking 2048 claims, such a system would take about 7.5 seconds just to verify the certificates. Our system can perform the same verification in less than 16 milliseconds. As another example, using Brands' credential is **O**(n) in exponentiations with respect to the number of claims [5]. Showing 2048 claims would require more than 2048 exponentiations. At a conservative 0.6 milliseconds per exponentiation, showing this credential would take about a second and a quarter – more than 80 times slower than our system. Of course, as mentioned earlier, Brands' credentials also provide stronger properties than we require, which necessitates their higher complexity.

Tests were performed on an Intel Core 2 Duo E6600 running at 2.4 GHz.

Operation	**Time (µ-seconds)**	**File size (bytes)**
Verify Tree (SHA-256)		
1 claim of 2048	1129	438
All 2048 claims	11789	31687
Verify Certificate		
RSA 1536 bit	3694	1074

Table 1. Time and space efficiency of hash tree and certificate.

6. CONCLUSION

Privacy is important both as a protective principle and as a security measure. Identity theft is a serious and widespread crime. The Federal Trade Commission reports that over a quarter of a million identity theft complaints were received in 2005, in addition to over 430,000 other fraud complaints. Internet-related complaints accounted for almost half of those [1]. Protecting personal information is vital to reducing identity theft. Limiting information disclosure does not require that accesses to service providers be completely anonymized. A user may repeatedly present the same credential or service provider specific claims to a service provider. This could facilitate many useful applications such as user-controlled services for monitoring usage of their credentials. In addition, for security sensitive applications and with appropriate safeguards in place, accumulated user information could be sent to the identity providers to enable auditing of credential use.

7. ACKNOWLEDGMENTS

The authors wish to thank Nortel Networks and the state of Georgia for their support, Alexandra Boldyreva for helpful discussions, and Ralph Merkle for inspiration.

8. REFERENCES

[1] Consumer Fraud and Identity Theft Complaint Data, Federal Trade Commission, 2006.

[2] Bayer, D., Haber, S. and Stornetta, W.S., "Improving the Efficiency and Reliability of Digital Time-Stamping". in *Sequences II: Methods in Communication, Security, and Computer Science*, (1993), Springer-Verlag, 329-334.

[3] Brands, S. Credentica - U-Prove SDK, Credentica Inc., 2007.

[4] Brands, S. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. MIT Press, Cambridge, MA, 2000.

[5] Brands, S., Demuynck, L. and Decker, B.D. A Practical System for Globally Revoking the Unlinkable Pseudonyms of Unknown Users *(Accepted to) 12th Australasian Conference on Information Security and Privacy*, Townsville, Queensland, Australia, 2007.

[6] Camenisch, J. and Herreweghen, E.V. Design and implementation of the idemix anonymous credential system *Proceedings of the 9th ACM conference on Computer and communications security*, ACM Press, Washington, DC, USA, 2002.

[7] Camenisch, J., Hohenberger, S. and Lysyanskaya, A. Compact E-Cash *Cryptology ePrint Archive*, 2005.

[8] Camenisch, J. and Lysyanskaya, A. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, Springer-Verlag, 2001.

[9] Cameron, K. The Laws of Identity *Microsoft Web Services Technical Articles*, 2005.

[10] Cates, J. Robust and Efficient Data Management for a Distributed Hash Table, Massachusetts Institute of Technology, 2003.

[11] Chaum, D. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM, 28* (10).

[12] Chaum, D. and Evertse, J.-H., A secure and privacy-protecting protocol for transmitting personal information between organizations. in *Advances in cryptology*, (Santa Barbara, California, 1987), Springer-Verlag, 118-167.

[13] Goldreich, O., Goldwasser, S. and Micali, S. How to construct random functions. *J. ACM, 33* (4). 792-807.

[14] Hardt, D. Identity 2.0, OSCON 2005, 2005.

[15] Johnson, R., Molnar, D., Song, D.X. and Wagner, D., Homomorphic signature schemes. in *Topics in Cryptology -- CTRSA 2002*, (Berlin, Germany, 2002), Springer-Verlag, 244-262.

[16] Merkle, R., A Certified Digital Signature. in *Advances in Cryptography*, (Santa Barbara, California, United States, 1989), Springer-Verlag New York, Inc., 218 - 238.

[17] Microsoft. Microsoft's Vision for an Identity Metasystem, 2005.