Rich Credentials for Remote Identity Proofing[*]

Karen Lewison and Francisco Corella

Revised July 10, 2017; first version October 15, 2016

Abstract

This is the first of a series of papers describing the results of a project whose goal was to identify five remote identity proofing solutions that can be used as alternatives to knowledge-based verification. This paper describes the first solution, which makes use of two new concepts that are described in detail in the paper. The first new concept is that of a *typed hash tree*, which can be used to represent a collection of key-value pairs and whose root label provides an omission-tolerant cryptographic checksum of the collection. The second concept is that of a *rich credential*, which is issued by an identity source to a subject and allows the subject to remotely present three verification factors to a verifier with whom the subject may have no prior relationship, including something that the user has (a private key), something that the user knows (a password), and something that the user "is" (one or more biometric features). A rich credential includes a typed hash tree, whose omission tolerance is used in the rich credential to provide selective disclosure of attributes and selective presentation of verification factors. In the first solution, a DMV issues a rich credential containing a facial image of the subject, which can also serve as the digital source of the printed photograph on a physical driver's license issued to the subject. The verifier performs face recognition with presentation attack detection by matching the facial image against the subject's face shown in an audio-visual stream of the subject reading prompted text, verifying the synchrony between the audio and video channel of the stream by tracking the subject's lips in the video stream and matching distinguishable visemes to phonemes in the audio stream.

[*] Patent pending.

Contents

1 Introduction 4

2 Parties and kinds of evidence in remote identity proofing 6

3 Verification without prior relationship 6
- 3.1 Biometric verification without prior relationship 6
- 3.2 Password Verification without Prior Relationship 8
- 3.3 Three-factor verification in remote identity proofing 9

4 Privacy features of rich credentials 9

5 Typed hash trees for omission-tolerant integrity protection 11
- 5.1 Overview 11
- 5.2 Formal definitions 14
  - 5.2.1 Typed hash trees 14
  - 5.2.2 Isomorphism of typed hash trees of the same kind . . . 16
  - 5.2.3 Subtrees, pruning and grafting 16
  - 5.2.4 Representation of a multiset of key-value pairs 18
- 5.3 Omission tolerance 18
- 5.4 Addition intolerance 18
  - 5.4.1 Linear description of a typed hash tree 18
  - 5.4.2 Stack-based label computation 19
  - 5.4.3 Typed hash trees with same root label 20
  - 5.4.4 Protection against attacks on a pseudo-random bit generator 23
- 5.5 An omission-tolerant cryptographic checksum 24

6 Rich credentials 24
- 6.1 Components of a rich credential 24
- 6.2 Typed hash tree of a rich credential 27
- 6.3 Node and label arrays 28
- 6.4 Peripheral subtrees 29
- 6.5 Attribute subtrees 31
- 6.6 Password subtree 31
- 6.7 Revocable biometric subtrees 33
  - 6.7.1 Revocable biometrics 33
  - 6.7.2 Revocable biometrics in a rich credential 34

6.8 Non-revocable biometric subtrees 35
6.9 Joint protection of the password and a biometric key against physical capture 36
6.10 Rich credential issuance protocol 37
6.10.1 Stage 1 37
6.10.2 Stage 2 38
6.11 Rich credential presentation and verification protocol 40
6.12 Proving knowledge of a private key over a secure connection in different kinds of cryptosystems 42
6.13 Installation of rich credentials in multiple devices 44

7 Solution 1: rich credential issued by a DMV 45

8 Security analysis 50
8.1 General threat model for all five solutions 50
8.1.1 Adversarial goals 50
8.1.2 Adversaries 51
8.2 Adversarial capabilities specific to Solution 1 52
8.3 Threats and mitigations pertaining to Solution 1 54
8.4 Security posture of Solution 1 58

9 Conclusion 60

10 Acknowledgments 61

List of Figures

1 A typed hash tree 12
2 Result of pruning a subtree 17
3 Components of a rich credential 25
4 Depth-first post order traversal of a typed hash tree 28
5 Peripheral subtrees 30
6 An attribute subtree 30
7 A password subtree 31
8 A revocable biometric subtree 34
9 A non-revocable biometric subtree 36
10 Issuance and presentation of a DMV rich credential 47

Main changes since the original version

- In Section 5, a leaf node that was the root of a subtree that has been pruned is now called a dangling node rather than a salt node, and salt nodes now have undistinguished types. (This was one of the alternatives discussed in Section 5.5 of the original version.)
- The informal conclusions drawn from the formal theorem of Section 5.4.3 have been greatly simplified.
- In the protocols of Section 6.12 for proving knowledge of a private key, the verifier's identity has been incorporated into the challenge to prevent a man-in-the-middle attack by a malicious verifier.
- In Section 7, the option of using a push notification to launch the native app that submits the audio-video stream has been removed as a precaution against a potential attack where the adversary captures the rich credential and causes the verifier to send a push notification to the subject, who uploads an audio-video stream that becomes part of a presentation of the stolen credential by the verifier.

1 Introduction

This is the first of a series of papers on the work that we have been doing in our research project on *Remote Identity Proofing* [1].

The goal of the project was to identify five remote identity proofing solutions that can be used as alternatives to knowledge-based verification. We have now identified five solutions that we believe to be secure and practical. This paper describes Solution 1, which relies on the new concept of a *rich credential* to provide up to three-factor verification of the identity of a subject to a verifier who has no prior relationship with the subject. It specifically proposes the use of a rich credential issued by a Department of Motor Vehicles (DMV), since DMVs are the most commonly used identity sources in the United States. It also includes materials generally relevant to remote identity proofing that will be referenced in the papers describing the other four solutions. The paper is organized as follows:

- Section 2 discusses the parties and types of evidence involved in remote identity proofing.

- Section 3 points out an essential difference between identity proofing and authentication, viz. the absence of prior relationship between the subject and the verifier in identity proofing, and discusses its technological implications.

- Section 4 discusses two privacy features supported by rich credentials, selective disclosure of attributes and selective presentation of verification factors.

- Section 5 defines the concept of a typed hash tree and proves that the root label of a typed hash tree can be used as an omission-tolerant cryptographic checksum. A rich credential is based on a typed hash tree that represents a collection of key-value pairs, some encoding attributes and some encoding verification data that supports the presentation of verification factors to a verifier with whom the subject has no prior relationship. (The word *key* in the term *key-value pair* has its database meaning rather than its cryptographic meaning.) The omission-tolerant integrity protection provided by a typed hash tree enables the selective disclosure of attributes and selective presentation of verification factors.

- Section 6 defines the concept of a rich credential and shows how it can be used to provide three-factor verification, based on something the subject has, something the subject knows and something the subject is, while protecting the subject's privacy by allowing for selective disclosure of attributes and selective presentation of verification factors.

- Section 7 describes the methods used in Solution 1 for issuing and presenting a rich credential.

- Section 8 provides a general threat model applicable to all five solutions, and a security analysis of Solution 1.

- Section 9 recapitulates.

2 Parties and kinds of evidence in remote identity proofing

In an identity proofing event, a *subject* presents identity evidence originating from an *identity source* to a *verifier*. As argued in [2], and consistently with [3] and [4], the evidence should originate from multiple sources.

In-person proofing often relies on a primary piece of evidence, which is often a picture ID, supplemented by secondary pieces of evidence that may not include a photograph. In the United States the primary piece of evidence may be, e.g., a driver's license issued by the DMV of one of the US states or territories or the District of Columbia, or a passport issued by the State Department. The secondary evidence may include a street address confirmation code and/or a proof of possession of a utility, financial or telecommunication account.

Remote identity proofing should also include multiple sources of evidence. Our five solutions are concerned with the remote presentation of primary evidence, to be combined with one or more additional pieces of secondary evidence. In Solution 1, described here, the primary evidence consists of a rich credential issued by the primary identity source, together with a password and one or more biometric samples that are submitted to the verifier along with the rich credential and verified against the rich credential.

3 Verification without prior relationship

An essential requirement of identity proofing, which sharply differentiates it from authentication, is that a verifier must be able to verify the identity of a subject with whom it has no prior relationship.

3.1 Biometric verification without prior relationship

There are four architectural configurations for biometric verification, but only one of them is suitable for our purposes, being able to provide a biometric verification factor usable for remote identity proofing in the United States:

1. In the biometric configuration most commonly used today, a user of a smart phone or other computing device presents a fingerprint or some other biometric sample to a sensor located on the device in order to

unlock the device or otherwise enable the use of a credential such as a certificate and its associated private key stored in the device. That credential could then be used in a remote identity proofing event where the user of the device is the subject. However this only provides one verification factor: possession of the device that contains the credential. The purpose of the biometric verification is to protect the credential stored in the device by requiring biometric authentication of the subject to the device before the credential can be used, not to provide an additional verification factor. The verifier may not be able to tell whether the subject has set biometric authentication and been careful to lock the device or the credential in the device when not in use.

2. In another configuration, the subject sends a biometric sample to the verifier, which matches it against *biometric verification data* derived from a *biometric enrollment sample* that the verifier has previously obtained from the subject. The biometric verification data may be a template derived from the enrollment sample, or the enrollment sample itself, or a randomized helper datum used in revocable biometric technology as described below in Section 6.7.1. To have obtained the enrollment sample the verifier must have had a prior relationship with the subject.

3. In yet another configuration, the subject sends a sample to the verifier, which performs a one-to-many match against a database of biometric verification data. In some countries there are databases containing biometric verification for all or most of the population, and a verifier may be able to match the sample presented by the subject against the database without having a prior relationship with the subject. In the United States, however, there is no nationwide biometric database that can be accessed for identity verification purposes. In each state, the Department of Motor Vehicles (DMV) has facial images of most of the state residents, but very few states make their facial image databases available for the purpose of identity verification.

4. In the one configuration that is suitable for our purposes, signed biometrics, an identity source performs biometric enrollment and provides the subject with signed biometric verification data. Signed biometrics are used in PIV cards, but only for controlling physical access to buildings. We use signed biometrics for remote identity proofing in

solutions 1, 2, 4 and 5. In Solution 1, described in this paper, the identity source acquires an enrollment sample from the subject (or multiple samples pertaining to multiple biometric modalities) and includes biometric verification data derived from the enrollment sample in a signed rich credential. The subject submits the credential and a biometric sample to the verifier, which verifies the signature, matches the sample against the verification data, and performs presentation attack detection on the submission of the sample.

3.2 Password Verification without Prior Relationship

When a password is used for authentication of a subject to a verifier, the verifier compares the password submitted by the user to a salted hash of the password stored in a database kept by the verifier. This requires a prior relationship between the subject and the verifier, in the course of which the subject has registered the password with the verifier and the verifier has generated a salt, computed the salted hash of the password, and stored the salt and the salted hash in the database.

A password can also be used to unlock a device or enable the use of a traditional cryptographic credential stored in the device. But, just as when a biometric sample is used to unlock a device or enable the use of a credential as discussed above, this does not provide an additional verification factor besides possession of the device.

In remote identity proofing we use a password differently. The subject chooses a password, generates a salt, hashes the password with the salt, and sends the salted hash to the identity source. A subject-controlled device retains the salt and never discloses it. The identity source issues a signed credential, where the signature covers the salted hash, with neither the salted hash nor the password being included in the credential as it is stored in the subject-controlled device. At proofing time, the device prompts the subject for the password, computes the hash of salt and the password, and sends the salted hash to the verifier, which embeds it into the credential and implicitly checks the password by verifying the signature on the credential with the embedded salted hash. (Equivalently, the salted hash may be embedded into the credential by the subject's device.)

We shall refer to the salted hash of the subject's password by the acronym *SHoSP*.

This method of using a password has strong security features:

- The verifier does not keep a password database.

- The verifier does not see the password.

- The verifier cannot use the salted hash to mount a dictionary attack against the password, because the verifier does not see the salt. By contrast, in a password database the salt and the salted hash are stored together, enabling an attacker who breaches the security of the database to test guesses of the password by hashing each guess with the salt and comparing the result to the salted hash.

- If the password is reused for multiple identity proofing credentials and for authentication at web sites, a verifier who sees a salted hash of the password used with one credential cannot use it in an attempt to impersonate the subject using a different credential, nor to authenticate as the subject at a web site. And a malicious web site operator who receives the password cannot use it in an attempt to impersonate the subject in a remote identity proofing event.

3.3 Three-factor verification in remote identity proofing

In authentication, the combination of three authentication factors including a password, a biometric, and a device containing a private key is often touted as the gold standard ("something you know, something you are, something you have"). This gold standard has not yet been achieved in remote identity proofing because the traditional ways of verifying a password and verifying a biometric sample require a prior relationship of the subject with the verifier. By obviating the need for a prior relationship, the rich credential of Solution 1 (and, similarly, the rich blockchain certificate of Solution 2 to be described in another paper) achieve the same gold standard in remote identity proofing.

4 Privacy features of rich credentials

Physical credentials are monolithic. When a physical credential such as a driver's license or a passport is presented as identity evidence to a verifier, the verifier sees and may record all the data in the credential, even though it may only need some of the data and may be required by law to only collect the data that it needs.

Some cryptographic credentials provide enhanced privacy by allowing the subject to select what attributes are presented to the verifier. This feature is known as *selective disclosure* of attributes. Anonymous credential cryptosystems, such as IBM's Idemix [5, 6] or Microsoft's U-Prove [7] provide selective disclosure, and furthermore provide *unlinkability*. Unlinkability refers to the inability of credential issuers and verifiers to use clues other than disclosed attributes to link the presentation of a credential to its issuance or to link different presentations of a credential to each other. (A discussion of different forms of unlinkability featured by different kinds of anonymous credentials can be found in [8].) But unlinkability does not matter for a credential that is used for remote identity proofing, since presentation of the credential is intended to identify the subject, and therefore the presentations of the credential can be linked to each other and to the issuance of the credential by the attributes that are intentionally disclosed by the subject.

A rich credential does not provide unlinkability, but it provides the privacy feature that matters in remote identity proofing: selective disclosure of attributes. At each identity proofing event that makes use of a rich credential, the subject and the verifier may negotiate which of the attributes asserted by the credentials are to be disclosed. And the cryptographic implementation of selective disclosure in a rich credential is very simple, the only cryptographic primitives used in a rich credential being digital signature, cryptographic hashing, and random-bit generation. By contrast, anonymous credentials use complex cryptographic primitives and, perhaps for that reason, are still in the experimental stage after more than 15 years of research and development.

A rich credential also provides another privacy feature: *selective presentation of verification factors*. A single rich credential issued by an identity source allows the subject to present to a verifier three kinds of verification factors: "something that the user has" (a device containing the credential), "something that the user knows" (a password, hashed with a secret salt), and "something that the user is" (one or more biometric samples, to be matched against biometric verification data for one or more biometric modalities in the credential). However, some identity proofing use cases may not warrant presentation of all the verification factors made available by a rich credential. For example, biometric verification may only be required in cases with high security requirements. Furthermore, when biometric verification is required and biometric verification data for multiple modalities is included in a credential, different verifiers may be equipped to verify different modalities. A rich credential allows the subject and the verifier to negotiate which verification factors will be used in each particular presentation. Allowing the subject to omit biometric verification when not required is an important privacy feature.

5 Typed hash trees for omission-tolerant integrity protection

This section defines a variation on the concept of a hash tree, which we call a *typed hash tree*, and proves that the root label of a typed hash tree can be used as an omission-tolerant cryptographic checksum. A rich credential, described below in Section 6, is based on a typed hash tree that has key-value pairs encoding attributes and verification data. The selective disclosure of attributes and selective presentation of verification factors featured by a rich credential are derived from the omission-tolerant integrity protection provided by a typed hash tree.

5.1 Overview

Consider a data structure that is used as a representation of a collection of key-value pairs. A cryptographic hash of an encoding of the data structure can be used as a checksum to ensure that the data structure has not been modified by an adversary. The checksum can be verified by comparing it to an original checksum supplied by the originator of the data structure, or by verifying a signature on the checksum.

A *typed hash tree* is a data structure that can be used to represent a collection of key-value pairs, and its root label can be used as a checksum of the key-value pairs present in the tree by comparing it to an original checksum or verifying a signature on the checksum. However, unlike a traditional cryptographic hash, the root label of a typed hash tree is an *omission-tolerant checksum*, in the sense that key-value pairs can be legitimately omitted from the tree without invalidating the checksum, but an adversary cannot add a key-value pair without invalidating the checksum.

The concept of a typed hash tree is illustrated in Figure 1. Informally, a typed hash tree is an ordered tree where each node has both a *type* and a

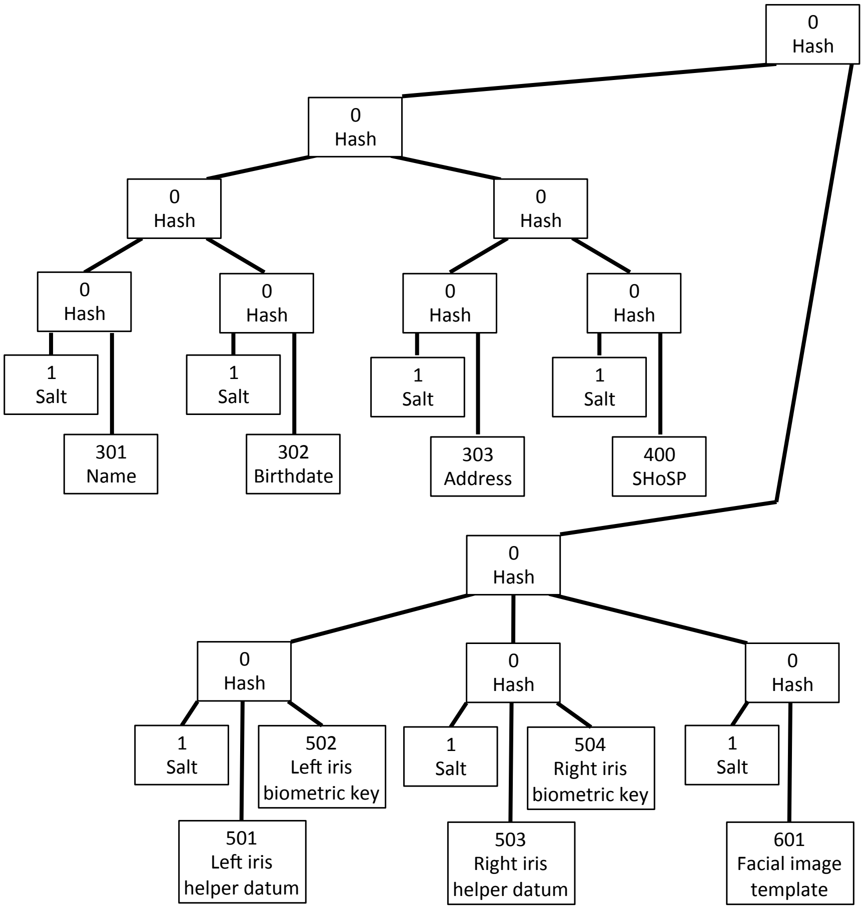

Figure 1: A typed hash tree. Each node has a type and a label. Each box in the figure illustrates a node, the number in the box being the type, the text in the box describing the kind of label. Type 0 is the distinguished type. Each internal node with type 0 is labeled by the hash of the types and labels of its children. Nodes with undistinguished types represent key-value pairs, the type being the key and the label being the value. SHoSP refers to the salted hash of the subject's password described in Section 3.2.

label, and the label of each internal node is a hash of the types and labels of its children. One distinguished type d, e.g. $d = 0$ if types are numeric, is shared by all the internal nodes and possibly some of the leaf nodes. Leaf nodes labeled by the distinguished type are called *dangling nodes*; a dangling node is shown below in Figure 2. Leaf nodes labeled by other types represent key-value pairs, the type and label of such a node playing the roles of key and value respectively. We shall view the collection of key-value pairs represented by the hash tree as a *multiset* because the same key-value pair may be represented by multiple nodes. Also, the same key may be paired with different values in different nodes.

The privacy features of a rich credential are implemented by *pruning* the typed hash tree of the credential. Given a typed hash tree X and subtree X' rooted at a node N of X, the subtree X' can be pruned from X without invalidating the root-label checksum by removing from X the nodes of X' other than N, causing N to become a *dangling node*. Pruning X' from X omits the key-value pairs represented by X' from the collection of key-value pairs represented by X.

A dangling node does not necessarily result from pruning a subtree; a dangling node with a random or pseudo-random label may be used to prevent a rich credential verifier from knowing whether a subtree has been pruned before credential presentation. For example, a rich credential profile for a particular use case may specify a collection of subtrees that may be included in the typed hash tree, but some of those subtrees may be optional and may be omitted if they are not applicable to a particular subject. A dangling node may be used in lieu of a subtree to hide the fact that the subtree is not applicable to a subject rather than being applicable but having been pruned.

Figure 1 shows several *salt nodes*. A salt node is labeled by a random or pseudo-random value. Salt nodes do not have a distinguished type, and need not all have the same type. In the figure they have type 1, as an example. The purpose of salt nodes is to protect key-value pairs omitted by pruning from a guessing attack by an adversary who has access to the pruned tree but not the original tree. Suppose the subtree X' pruned from the typed hash tree X includes a leaf node N' with type $t \neq d$ and label l, representing the key-value pair (t, l). The root N of the X' remains in the pruned tree, and the label of N can be computed from the types and labels of the leaf nodes of X', including t and l. Therefore the adversary may attempt to test guesses of the omitted key-value pair (t, l) (or guesses of l if t is known) against the known label of N. But if N' has a sibling labeled by a random salt, the adversary has to make simultaneous guesses of (t, l) and the random salt, and will have a negligible probability of finding the correct guess if the random salt has high entropy.

While it is possible to remove key-value pairs from a typed hash tree X without changing its root label by pruning one or more subtrees, we shall see below that, if the dangling nodes of X are have random labels, it is deemed infeasible to add key-value pairs to X without updating its root label. This justifies the statement that the root label of a typed hash tree is an *omission-tolerant cryptographic checksum* of the collection of key-value pairs represented by the tree.

5.2 Formal definitions

5.2.1 Typed hash trees

Formally, a *kind* of typed hash tree is a tuple $\mathcal{K} = (T, d, L, h, f)$, where:

- T is a set of elements called *types*.
- $d \in T$ is a *distinguished type*.
- L is a family of sets $(L(t))_{t \in T}$.
- h is a cryptographic hash function whose codomain is $L(d)$.
- f is an injective function that maps sequences of pairs (t, l), where $t \in T$ and $l \in L(t)$, to elements of the domain of h.

Given such a kind $\mathcal{K} = (T, d, L, h, f)$, a *typed hash tree of kind* $\mathcal{K}$ is a tuple $X = (\mathcal{K}, \mathcal{N}, \mathcal{T}, \mathcal{L}, \mathcal{F})$ where:

- $\mathcal{N}$ is a finite set whose elements are called the *nodes* of the tree.
- $\mathcal{T}$ is a function from $\mathcal{N}$ to T that maps each $N \in \mathcal{N}$ to an element $t \in T$ called *the type of* N.
- $\mathcal{L}$ is a function from $\mathcal{N}$ to $\bigcup_{t \in T} L(t)$ that maps each $N \in \mathcal{N}$ to an element of $L(\mathcal{T}(N))$ called *the label of* N.
- $\mathcal{F}$ is a function that maps each node $N \in \mathcal{N}$ to a sequence without repetitions, the nodes in the sequence being called the *children* of N, nodes without children being called *leaf nodes* and nodes with children being called *internal nodes*, such that:

-
  - Every node N is a child of at most one other node, which is called the parent of N.
  - The parent-child relation forms a directed acyclic graph.
  - Exactly one node, called the *root* of the tree, has no parent.

- Every internal node has type d. A leaf node may have type d, in which case it is called a *dangling node.*

- The type of the root node is d. The root node may be either an internal node or a dangling node.

- Every internal node N has as its label

  $$l = h(f(s)),$$

  where s is the sequence of type-label pairs of the children of s; i.e., if $\mathcal{F}(N) = (N_i')_{0\leq i<n}$, $n > 0$, s is the sequence

  $$s = ((\mathcal{T}(N_i'), \mathcal{L}(N_i')))_{0\leq i<n} \,.$$

Typically, h maps bit strings of arbitrary length to bit strings of some fixed length k such as 256, 384 or 512. Then, according to the above definitions, the label of an internal node N is a k-bit string that is computed in two steps:

1. First, the function f, which performs the role of a one-to-one encoding, is applied to the sequence s of type-label pairs of the children of N, producing a bit string that we shall call the *prelabel* of N.

2. Then the hash function h is applied to that prelabel to compute the label of N.

For example, if N has three children with types t_1, t_2, t_3 and labels l_1, l_2, l_3, then the sequence of type-label pairs is

$$s = [(t_1, l_1), (t_2, l_2), (t_3, l_3)],$$

the prelabel of N is

$$f(s) = f([(t_1, l_1), (t_2, l_2), (t_3, l_3)]),$$

and the label of N is

$$h(f(s)) = h(f([(t_1, l_1), (t_2, l_2), (t_3, l_3)]).$$

Note. We use square brackets to denote sequences, e.g. $[\,]$ denotes $\emptyset$, $[a]$ denotes $\{(0,a)\}$, $[a,b]$ denotes $\{(0,a),(1,b)\}$, $[a,b,c]$ denotes $\{(0,a),(1,b),(2,c)\}$, etc.

5.2.2 Isomorphism of typed hash trees of the same kind

We say that two trees of the same kind are isomorphic if there exists a bijection between their sets of nodes such that corresponding nodes have same type, same label, and sequences of corresponding children.

5.2.3 Subtrees, pruning and grafting

Given a typed hash tree $X = (\mathcal{K}, \mathcal{N}, \mathcal{T}, \mathcal{L}, \mathcal{F})$ and a node N of X, the set of *descendents* of N in X (N not being considered a descendent of itself), is the smallest set S that contains the children of N and, for every $N' \in S$, also contains the children of N'. The *subtree* of X rooted at N is the typed hash tree $X' = (\mathcal{K}, \mathcal{N}', \mathcal{T}', \mathcal{L}', \mathcal{F}')$ where $\mathcal{N}'$ is the union of $\{N\}$ and the set of descendents of N in X, and $\mathcal{T}'$, $\mathcal{L}'$ and $\mathcal{F}'$ are the restrictions of the functions $\mathcal{T}$, $\mathcal{L}$ and $\mathcal{F}$ to the domain $\mathcal{N}'$.

Given a type hash tree $X = (\mathcal{K}, \mathcal{N}, \mathcal{T}, \mathcal{L}, \mathcal{F})$ and a subtree X' of X rooted at node N, the result of *pruning X' from X* is the typed hash tree $X'' = (\mathcal{K}, \mathcal{N}'', \mathcal{T}'', \mathcal{L}'', \mathcal{F}'')$, where $\mathcal{N}''$ is the set of nodes of X that are not descendents of N, $\mathcal{T}''$ and $\mathcal{L}''$ are the restrictions of $\mathcal{T}$ and $\mathcal{L}$ to $\mathcal{N}''$, and $\mathcal{F}''$ is the function that maps N to $\emptyset$ and every node N' of X that is not a node of X' to $\mathcal{F}(N')$. Figure 2 illustrates the result of pruning the leftmost three-node subtree from the typed hash tree of Figure 1.

A *pruned derivative* of X is a tree derived from X by pruning zero or more subtrees. (X is a pruned derivative of itself.)

Given a typed hash tree $X = (\mathcal{K}, \mathcal{N}, \mathcal{T}, \mathcal{L}, \mathcal{F})$ with a dangling node S, and a typed hash tree $X' = (\mathcal{K}, \mathcal{N}', \mathcal{T}', \mathcal{L}', \mathcal{F}')$ of same kind $\mathcal{K}$ as X, with root S, such that $\mathcal{N} \cap \mathcal{N}' = \{S\}$ and $\mathcal{L}(S) = \mathcal{L}'(S)$, the typed hash tree X'' derived by *grafting* X' onto X is defined as the typed hash tree $X'' = (\mathcal{K}, \mathcal{N}'', \mathcal{T}'', \mathcal{L}'', \mathcal{F}'')$ where $\mathcal{N}'' = \mathcal{N} \cup \mathcal{N}'$, $\mathcal{T}'' = \mathcal{T} \cup \mathcal{T}'$, $\mathcal{L}'' = \mathcal{L} \cup \mathcal{L}'$, and $\mathcal{F}''$ is the function with domain $\mathcal{N}''$ that maps every node N of X other than S to $\mathcal{F}(N)$ and every node N' of X' to $\mathcal{F}'(N')$.

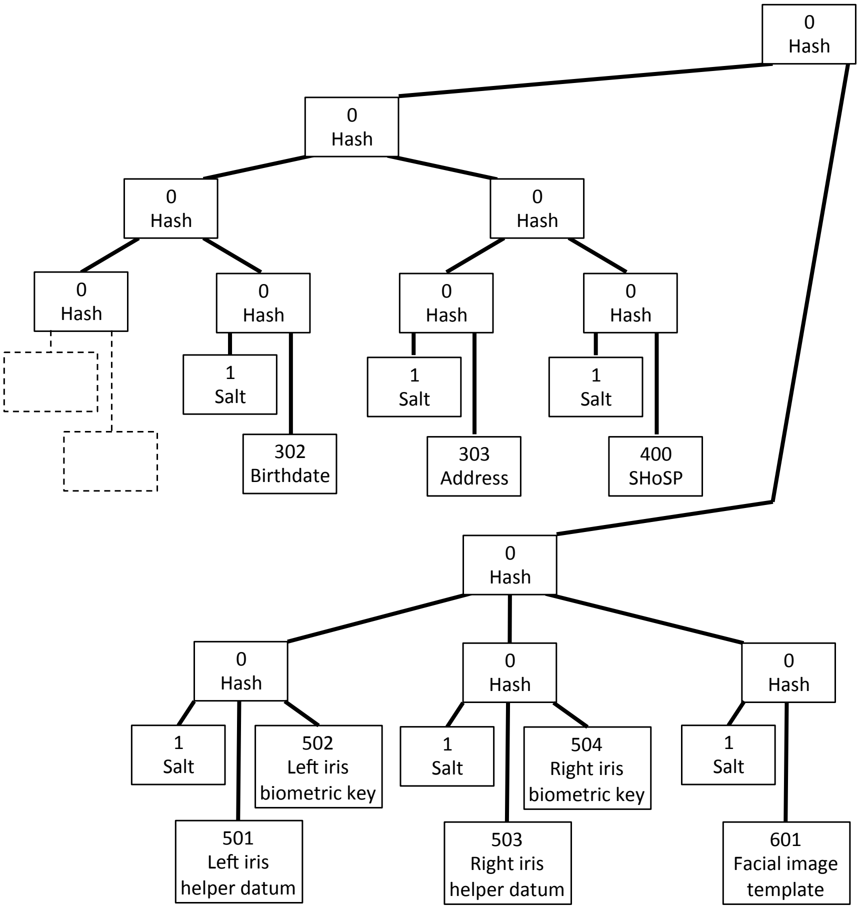

Figure 2: Result of pruning the leftmost three-node subtree from the typed hash tree of Figure 1. Dashed-lines are used to illustrate the nodes that have been removed. The root of the subtree becomes a dangling node, but its type and its label do not change.

5.2.4 Representation of a multiset of key-value pairs

A typed hash tree $X = (\mathcal{K}, \mathcal{N}, \mathcal{T}, \mathcal{L}, \mathcal{F})$ of kind $\mathcal{K} = (T, d, L, h, f)$ is a representation of a multiset M of key-value pairs defined as follows:

- A leaf node N of type $t \neq d$ with label $l \in L(t)$ represents the key-value pair (t, l), where t is the key and l is the value.
- The elements of M are the key-value pairs that are thus represented by the non-dangling leaf nodes of X, the multiplicity of each such pair in M being the number of nodes of X that represent the pair.

5.3 Omission tolerance

The following facts follow directly from the above definitions and show that it is possible to omit key-value pairs from a typed hash tree without invalidating the root label of the tree.

Fact 1. *The multiset of key-value pairs represented by a pruned derivative X' of a typed hash tree X is a submultiset of the multiset of key-value pairs represented by X; it is a strict submultiset if $X' \neq X$.*

Fact 2. *A pruned derivative of a typed hash tree X has the same root label as X.*

Fact 2 can be viewed as stating that the root label of a typed hash tree, if used as a checksum of the key-value pairs represented by the tree, has *omission tolerance*.

5.4 Addition intolerance

The concepts of *linear description* and *stack-based label computation (SBLC)* of a typed hash tree, defined below in sections 5.4.1 and 5.4.2, are needed in Section 5.4.3 for the proof of Theorem 1.

5.4.1 Linear description of a typed hash tree

The linear description of a typed hash tree of kind (T, d, L, h, f) is a sequence whose entries correspond to the nodes of the tree arranged in depth-first post order, the entry corresponding to a node N being itself a sequence consisting of the type of N, followed by number of children of N and, if N is a leaf node, by the label of N. Thus if N is a leaf node with type t and label l, the corresponding entry of the linear description is $[t, 0, l]$, and if N is an internal node with n children, the corresponding entry is $[d, n]$.

The linear description of a typed hash tree X of kind (T, d, L, h, f) can be used to define another typed hash tree Y of same kind as X whose nodes are the indices of entries in the description, as follows. If i is the index of an entry $[t, 0, l]$, i is a leaf node of Y with type t and label l. If i is the index of an entry $[d, n]$, i is an internal node of Y with type d, the sequence of its children is $[i - n, i - n - 1, \ldots, i - 1]$, and its label is recursively defined as $h(f(s))$, where s is the sequence of the type-label pairs of its children. The bijection between the nodes of X and their positions in the depth-first post order traversal of X, which are the positions of their corresponding entries in the linear description X, is then an isomorphism between X and Y.

Hence we can say that two trees of same kind that have the same linear description are isomorphic, and, since the converse is trivially true, that two trees of same kind are isomorphic iff they have the same linear description.

5.4.2 Stack-based label computation

The labels of the internal nodes of a typed hash tree, up to the root label, can be computed from the types and labels of the leaf nodes. One particular method for performing this computation traverses the nodes of the tree in depth-first post order and keeps track of intermediate results in a stack. We refer to a computation performed by this method as a *stack-based label computation (SBLC)*. The proof of Theorem 1 makes use of a formal specification of the SBLC of a typed hash tree, which we provide in this section.

For our purposes, a stack is a just a mathematical sequence; the top of the stack is the last entry in the sequence; pushing to the stack means appending to the sequence; popping from the stack means removing from the end of the sequence.

The SBLC of a typed hash tree X of kind (T, d, L, h, f) consists of a sequence of steps triggered by the nodes of X arranged in depth-first post order. Each step takes the computation from one stage to a next stage. Each stage is a pair (S, R), where S is a stack consisting of pairs (t, l) where $t \in T$ and $l \in L(t)$, and R is a suffix of the linear description of X (R is mnemonic for "the *R*est of the description").

In the initial stage (S_0, R_0), S_0 is the empty stack and R_0 is the entire linear description.

Each node N of X triggers a step that takes the computation from stage (S, R) to stage (S', R') as follows:

- R' is derived from R by removing the first entry of R.
- If N is a leaf node with type t and label l, S' is derived from S by pushing (t, l) to S. We refer to a step where this is the case as a *push step.*
- If N is an internal node with n children $(n > 0)$, S' is derived from S by popping n entries and pushing the entry $(d, h(f(s)))$, where s is the sequence of entries popped. We refer to a step where this is the case as a *hash step.*

In the final stage of the computation, the stack contains a single entry (d, r), where r is the root label of X, and the suffix of the linear description is empty.

Since each step removes the first entry from R, the step triggered by node N applies to a stage where the first entry of R is the entry of the linear description that corresponds to N. The computation is entirely defined by its initial stage, and could be specified by referring to entries of the linear description rather than nodes of X. However, it will be convenient for the proof of Theorem 1 to refer to nodes of X triggering steps of the computation.

5.4.3 Typed hash trees with same root label

Theorem 1. *Let X and Y be two typed hash trees of the same kind. If X and Y have the same root label, then either (i) Y is isomorphic to a pruned derivative of X (which may be X itself), or (ii) the label of an internal node of Y is equal to the label of a dangling node of X, or (iii) an internal node of Y has the same label as an internal node of X but a different prelabel.*

Proof. Let X and Y be two typed hash trees of same kind (T, d, L, h, f) that have the same root label.

Let C be the SBLC of Y. In the last stage of C the stack consists of one pair (d, r), where r is the root label of Y, and the linear description is empty. Since r is also the root label of X, the last stage of C is the same as the last stage in the SBLC of X. Hence, since X is a pruned derivative of itself, the set of stages of C that are also stages of the SBLC of a pruned derivative of X is not empty.

Let $E = (S, R)$ be the earliest stage of C that is also a stage of the SBLC of a pruned derivative of X, let X' be a minimally pruned such derivative, and let C' be the SBLC of X'. (By X' being a "minimally pruned such derivative" we mean that X' is a pruned derivative of X having E as a stage of its SBLC, and is not a pruned derivative of another pruned derivative of X having E as a stage of its SBLC.)

In an SBLC, the stack is only empty in the initial stage of the computation. Therefore if S is empty, E is the initial stage in both C and C', and R is the entire linear description of both Y and X'. Having the same linear description, Y and X' are isomorphic, and Y satisfies condition (i) of the theorem.

If S is not empty, let (t, l) be the entry at the top of S, and consider the steps leading to stage E in C and C', which we shall call "the C step" and "the C' step" respectively. Let N be the node of Y that triggers the C step, and N' the node of X' that triggers the C' step. Both N and N' have type t and label l, but may differ in the number of their children when their type is d.

The C step and the C' step cannot both be push steps, because then E would be preceded in both C and C' by the same stage $E' = (S', R')$, with S' being derived from S by popping one entry, and R' from R by prepending the entry $[t, 0, l]$. This would contradict the definition of E as the earliest stage of C that is also a stage of the SBLC of a pruned derivative of X, E' being an earlier stage of C that is also a stage of C'.

Only a push step can put an entry (t, l) with t different from d at the top of the stack. Hence, since either the C step or the C' step is not a push step, we must have $t = d$.

The C' step cannot be a hash step if the C step is a push step. If that were the case, let X'' be the tree obtained by pruning the subtree of X' rooted at N', and let C'' be the SBLC of X''. The SBLCs of X' and X'' would be identical following the step triggered by N' (which is a node of X'' as well as X', and triggers a hash step in X' and a push step in X''). Therefore E would also be a stage of the SBLC C'' of X'', and it would be preceded in C and C'' by the same stage $E' = (S', R')$, S' being derived from S by popping one entry, and R' from R by prepending the entry $[t, 0, l]$. This would contradict the definition of E as the earliest stage of C that is also a stage of the SBLC of a pruned derivative of X, E' being an earlier stage of

C that is also a stage of C'''.

This leaves two possibilities. The C' step may be a push step while the C step is a hash step, or the C' step and the C step may both be hash steps.

Consider first the case where the C' step is a push step while the C step is a hash step. Then N is an internal node of C, $t = d$, and N' is a leaf node with type d, i.e. a dangling node, of X'. But N' must also be a dangling node of X, for otherwise the tree X''' derived from X' by grafting the subtree of X rooted at N' would be a pruned derivative of X having E as a stage of its SBLC, and X' would be a pruned derivative of X''', contradicting the minimality of X'. Therefore N' is a dangling node of X having the same label l as the internal node N of Y, hence Y satisfies condition (ii) of the theorem.

Now consider the case where the C step and the C' step are both hash steps. Then N and N' are internal nodes of Y and X' with same label l. If s and s' are the sequences of the type-label pairs of the children of N in Y and N' in X' respectively, we have $l = h(f(s)) = h(f(s'))$. Let S' by the result of popping the top entry of S. In the stage of C that precedes E, the stack consists of the entries of S' followed by those of s, and the suffix of the linear description has $[d, n]$ as its first entry, where n is the length of s, followed by the entries of R. Similarly, in the stage of C' that precedes E, the stack consists of the entries of S' followed by those of s', and the suffix of the linear description consists of $[d, n']$ followed by the entries of R, where n' is the length of s'. The sequences s and s' cannot be identical, because then the stages of C and C' preceding E would be identical, contradicting the definition of E as the earliest stage of C that is also a stage of the SBLC of a pruned derivative of X (such as X'). Therefore, since f is one-to-one, the prelabels $f(s)$ and $f(s')$ of the internal node N of Y and the internal node N' of X' are not the same. Hence N is an internal node of Y having the same label l as the internal node N' of X' but a different prelabel. But it follows from the fact that X' is a pruned derivative of X that N' is also a node of X and has the same prelabel in X as in X'. Thus an internal node of Y has the same label as an internal node of X but a different prelabel, and Y satisfies condition (iii) of the theorem. $\square$

Cryptographic hash functions are deemed to have certain security properties, including collision resistance and preimage resistance. Informally, a hash function h is said to have *collision resistance* if it is infeasible to find a collision, i.e. a pair (x, x') such that $h(x) = h(x')$, and to have *preimage re-*

sistance if, given y chosen at random with a uniform probability distribution over the codomain of h, it is infeasible to find x such that $y = h(x)$. The term "infeasible" is informal in these definitions, and is difficult to formalize when defining the security of unkeyed hash functions [9, end of Section 5.1.1].

Since the hash function h of a typed hash tree X is a cryptographic hash function and is therefore deemed to have collision and preimage resistance, it follows from Theorem 1 that, if the labels of the dangling nodes of X, if any, are chosen uniformly at random from the codomain of h, it is infeasible for an adversary to construct a typed hash tree Y of same kind as X that has the same root label as X but whose key-value pairs are not a submultiset of those of X. Indeed, if X and Y have the same root label, then X and Y have to satisfy one of conditions (i), (ii) or (iii) of Theorem 1. But by Fact 1, if the key-value pairs of Y are not a submultiset of those of X, Y is not a pruned derivative of X and condition (i) is not satisfied. And, since the label of an internal node is the hash of its prelabel, if conditions (ii) or (iii) were satisfied, the adversary would have breached the preimage resistance or collision resistance of h, respectively, which is deemed infeasible.

Thus Theorem 1 implies that the root label of a typed hash tree has *addition intolerance* with respect to the key-value pairs represented by the tree, provided that the labels of the dangling nodes, if any, are chosen uniformly at random from the codomain of the hash function of the tree.

5.4.4 Protection against attacks on a pseudo-random bit generator

The proviso of the above addition-intolerance result can be satisfied by using a pseudo-random bit generator to generate the labels of the dangling nodes, seeded with at least k bits of entropy if the codomain of h is the interval $[0, 2^k)$.

In practice it may be difficult to be sure that the entropy requirement is satisfied. In particular it may not be satisfied if the generator has been compromised by an adversary. To mitigate the risk that the generator has been compromised or may not have enough entropy for some other reason, it may be prudent to apply a hash function h' other then h to the output of the generator.

Doing so is a mitigation because different members of a family of hash functions, such as, e.g. SHA-256, SHA-386 and SHA-512, are designed to avoid *cross-collisions* between them, i.e. pairs (x, x') such that $h(x) = h'(x')$

where h and h' are different members of the family. It should also be reasonable to assume that it is infeasible to find cross-collisions between members of different families of hash functions. Therefore if the label y of a dangling node of X is the output of a hash function h', it should be infeasible for the adversary constructing Y to create an internal node of Y whose label is the output of the hash function h of the typed hash trees X and Y.

5.5 An omission-tolerant cryptographic checksum

The omission-tolerance result of section 5.3 and the addition-intolerance result of section 5.4.3 can be combined to state that the root label of a typed hash tree whose dangling nodes (if any) have random labels is an *omission-tolerant cryptographic checksum* of the collection of key-value pairs represented by the tree.

6 Rich credentials

A rich credential is based on a *typed hash tree*, which is pruned as desired when the credential is presented, to allow for selective disclosure of attributes and selective presentation of verification factors. State transitions are described in sections 6.5–6.8.

6.1 Components of a rich credential

A rich credential comprises the following components, as illustrated in Figure 3:

1. A *private key*, which is a component of a key pair. The key pair is generated in the device that carries the credential and the private key never leaves the device. We shall refer to the device that carries the credential as *the subject's device* or, simply, the device. Possession of the subject's device that contains the private key is one of the verification factors provided by the rich credential. The key pair may pertain to any kind of public key cryptosystem, such as a signature cryptosystem, an encryption cryptosystem or a key exchange cryptosystem.

2. A *secret salt* that is used to compute the salted hash of the subject's password (SHoSP) of Section 3.2. This secret salt and the hashing used

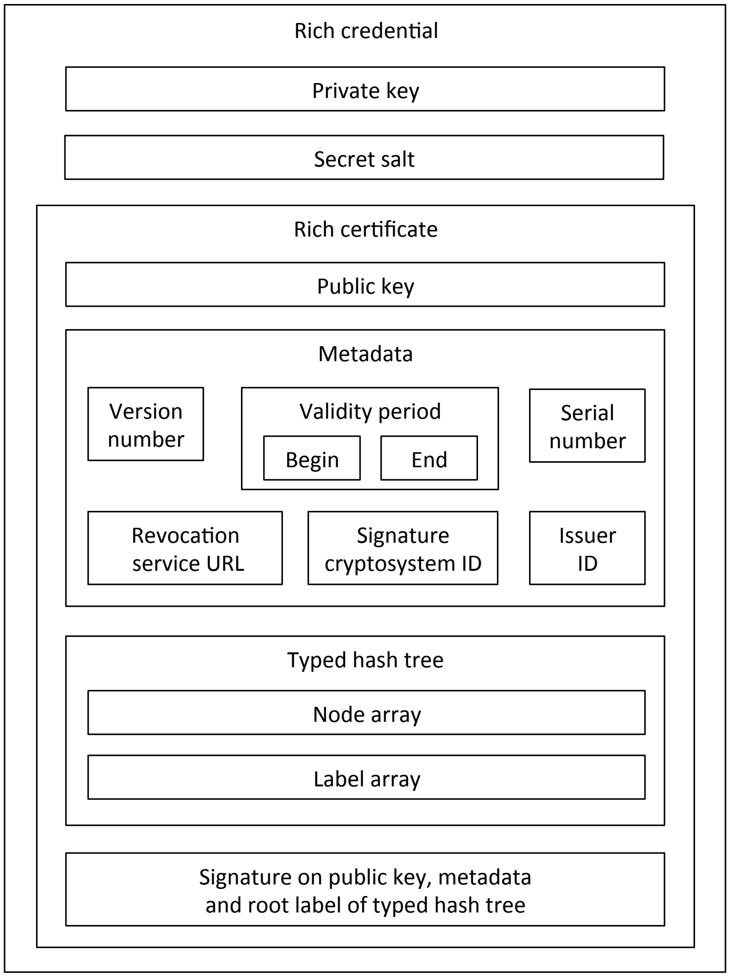

Figure 3: Components of a rich credential

to compute the SHoSP are unrelated to the salts and the hashing used in a typed hash tree as described in Section 5. Like the private key, the secret salt is generated in the subject's device and never leaves the device.

3. A *rich certificate* comprising:

   (a) The public key component of the key pair.

   (b) Metadata like the metadata found in an ordinary public key certificate, including:

      i. A version number, which references a specification of a particular format of a rich credential, thus allowing the details of the format to evolve over time.

      ii. A validity period.

      iii. A serial number.

      iv. A URL of a service that provides revocation information, such as a signed or unsigned list of serial numbers of revoked certificates, or the revocation status of a particular certificate identified by its serial number.

      v. A signature cryptosystem identifier that identifies the algorithm used to compute the signature included in the certificate, and the corresponding algorithm to be used for verifying it.

      vi. An organization identifier that identifies the issuer of the rich credential.

   (c) A description of the typed hash tree of the credential, comprising a *node array* and a *label array*, which mutate as the credential transitions from one state to another.

   (d) A *signature* by the credential issuer on the public key, the metadata, and the root label of the typed hash tree. Crucially, the signature covers the root label of the tree, rather than the node array and the label array included as components of the certificate. This allows the node array and the label array to change without invalidating the signature as the credential transitions from one state to another.

The rich certificate is a mutable data structure that goes through four states:

1. The *issuance state*, which is the state of the certificate after the issuer has computed the signature but before the certificate has been modified for transmission to the subject.

2. The *storage state*, i.e. the state in which it is transmitted by the issuer to the subject and stored in the subject's device. The salted hash of the password (SHoSP), and any biometric keys (see Section 6.7.1), are not present in the storage state.

3. The *presentation state*, i.e. the state in which the credential is transmitted by the subject to the verifier. Attributes that are not disclosed and verification data for verification factors that are not presented are omitted from the presentation state.

4. The *verification state*, i.e. the state in which the credential signature is verified. The verifier may add data such as the SHoSP or a biometric key to the presentation state before verifying the verifying the credential.

6.2 Typed hash tree of a rich credential

An example of a typed hash tree of a rich credential was shown above in Figure 1. The same example is shown again in Figure 4, with added node numbers indicating the position of each node in the depth-first post order traversal of the tree. Each box illustrates a node and the lines between boxes represent the parent-child relation. In each box the first line is the type of the node and the second line informally describes the label, while the number next to the box is the node number.

For example, node no. 1 has type 301 and its label is the name of the subject (e.g. represented as an UTF8 string); it represents the name attribute of the subject, encoded as a key-value pair, where 301 is a numeric code for the subject-name attribute. Such codes are defined by a specification of the credential, referenced by the version number in the metadata component of the rich certificate.

Recall that internal nodes and salt nodes are labeled by the distinguished type. In the example, the distinguished type of the typed hash tree if $d = 0$.

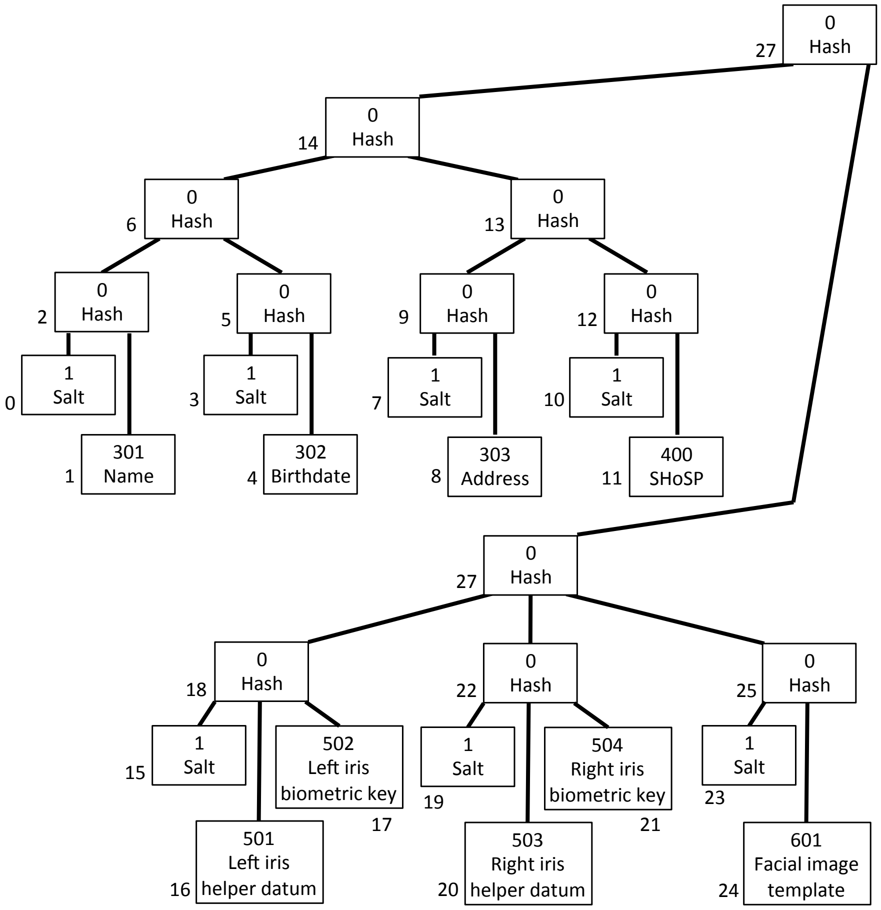

Figure 4: Depth-first post order traversal of a typed hash tree

Node no. 6 is an internal node, with type 0, labeled by a hash of the types and labels of its children, which are the nodes numbered 2 and 5. Recall also that salt nodes are leaf nodes with random labels and undistinguished types. In the example of Figure 4 they all have type 1.

6.3 Node and label arrays

The node array and label array components of the rich certificate provide a description of the typed hash tree of the credential.

The node array has entries for the nodes of the tree, listed in depth-first post order. Each entry is itself a two-entry array containing the type of the node and the number of children of the node, but not the label. As an example, here is the node array of the typed hash tree of Figure 4:

```
[ [1,0], [301,0], [0,2], [1,0], [302,0], [0,2], [0,2], [1,0], [303,0],
  [0,2], [1,0], [400,0], [0,2], [0,2], [0,2], [1,0], [501,0], [502,0],
  [0,3], [1,0], [503,0], [504,0], [0,3], [1,0], [601,0], [0,2], [0,3],
  [0,2] ]
```

The label array is a sparse array containing the labels of some of the nodes of the trees, each at the same position as the entry for the node in the node array, i.e. at the position of the node in the first-order depth first traversal. Different labels are included in different states of the certificate. All the labels are included in the issuance state, and all the labels of the possibly pruned tree are included in the verification state, but the labels of most internal nodes are excluded in the storage and presentation states, as discussed in Section 6.9 and specified in sections 6.10 and 6.11.

6.4 Peripheral subtrees

In the typed hash tree of a rich credential, every internal node is either a *peripheral node*, all of whose children are leaf nodes, or a *central node*, all of whose children are internal nodes. A *peripheral subtree* is a subtree rooted at a peripheral node. For example, the typed hash tree shown in figures 1 and 4 has seven peripheral subtrees, shown in Figure 5.

The typed hash tree of a rich credential may include four categories of peripheral subtrees:

- Attribute subtrees, one for each attribute of the subject asserted by the issuer.

- A password subtree, enabling the use of a password as a verification factor.

- Revocable biometric subtrees, one for each revocable biometric modality for which the credential asserts verification data.

- Non-revocable biometric subtrees, one for each non-revocable biometric modality for which the credential asserts verification data.

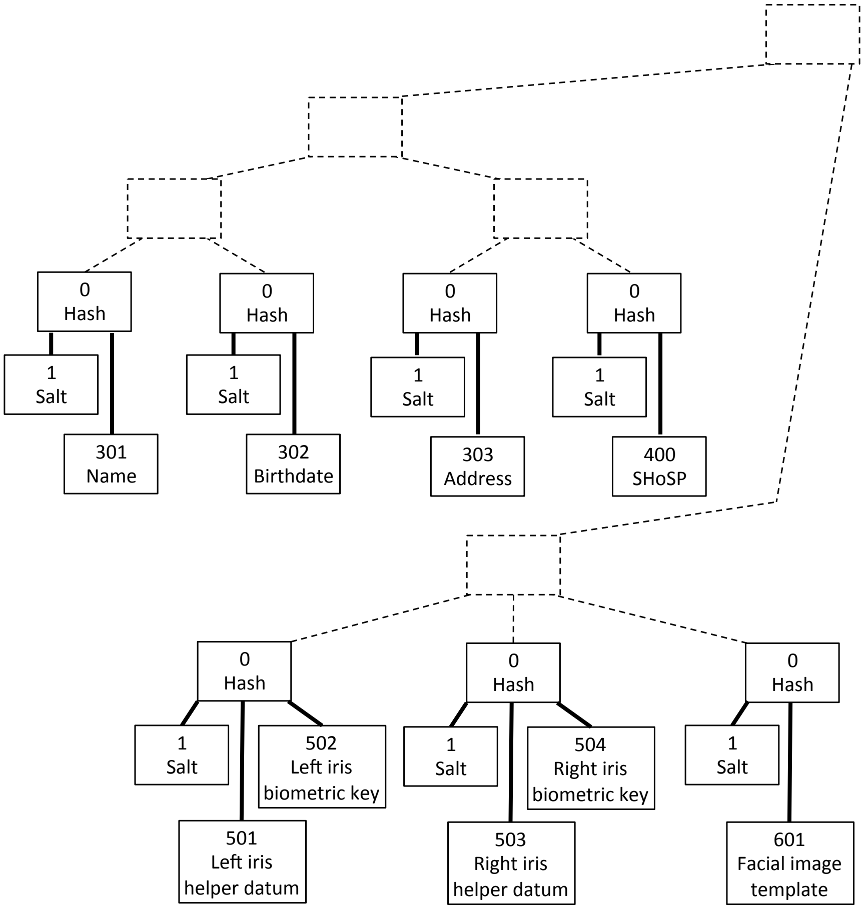

Figure 5: The seven peripheral subtrees of the typed hash tree of figures 1 and 4.

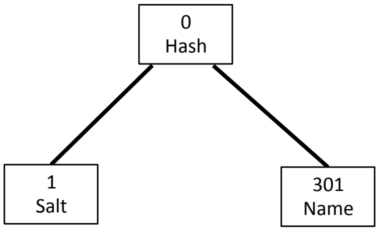

Figure 6: An attribute subtree

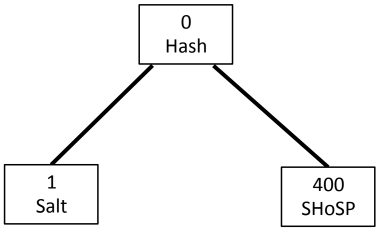

Figure 7: A password subtree

6.5 Attribute subtrees

Figure 6 shows an example of an *attribute subtree* in the issuance state. It has a peripheral node and two leaf nodes: a salt node and an attribute node. The label of the salt node is a random salt generated by the issuer, which we call the *sibling salt*. The attribute node represents an attribute encoded as a key-value pair. In the example it represents the subject-name attribute, as discussed above in Section 6.2.

In the storage state the label array includes the labels of the leaf nodes, but there is no need to store the label of the peripheral node.

If the attribute is omitted in a credential presentation, the label of the peripheral node is computed from the types and labels of the two leaf nodes and added to the label array. Then the attribute subtree is pruned from the typed hash tree in the presentation state, by removing the entries for the leaf nodes from the node array, and adjusting the indices of the entries in the label array accordingly. The peripheral node becomes a dangling node.

If the attribute is not omitted, the attribute subtree is not altered in the transitions from the storage state to the presentation and verification states.

The purpose of the sibling salt is to prevent the attribute from being guessed by the verifier when it is not intentionally disclosed in a presentation of the credential. Without the sibling salt, the verifier could test guesses of the attribute against the label of the dangling node, which would be a hash of the attribute, and would have no difficulty in guessing it, as the values of attributes such as name or birthdate have little entropy.

6.6 Password subtree

Figure 7 shows an example of a *password subtree* in the issuance state. It has a peripheral node and two leaf nodes: a salt node and a password node. The salt node has the undistinguished type 1, and its label is a sibling random salt generated by the issuer. In the example, the type of the password node is 400. The password node is labeled by the SHoSP. Recall that the SHoSP is the hash of the password with the secret salt, which is a component of the rich credential that is generated by the subject's device and never leaves the device. We stress that the secret salt is not one of the salts that label nodes of the typed hash tree.

At issuance time, the subject's device asks the subject to choose the credential password, generates the secret salt and computes the SHoSP, which it sends to the issuer. The issuer uses the SHoSP in the computation of the root label of the typed hash tree and the signature that is included in the certificate. But the issuer removes the SHoSP from the label array as it transitions the certificate to the storage state before delivering it to the subject's device. Thus in the storage state the label array includes the random salt but not the SHoSP. It also includes the label of the peripheral node, except as discussed below in Section 6.9 and specified in section 6.10.

A rich credential features selective presentation of verification factors, and knowledge of the password is one of the verification factors that may be omitted from a presentation. If it is omitted, the password subtree is pruned from the typed hash tree in the presentation state, by removing the entries for the leaf nodes from the node array, and adjusting the indices of the entries in the label array accordingly. The peripheral node becomes a dangling node.

If the password verification factor is not omitted, the subject's device removes the label of the peripheral node from the label array in the transition from the storage state to the presentation state; this preempts a possible bug in the verifier code, which might otherwise fail to verify that the label of the peripheral node is the hash of the types and labels of its children. The subject's device prompts the subject for the password, computes the SHoSP, and sends the SHoSP to the verifier along with the certificate in the presentation state. The verifier adds the SHoSP to the label array as it transitions the certificate from the presentation state to the verification state. (Equivalently, the SHoSP may be added to the label array by the subject's device as part of the transition to the presentation state.)

As in an attribute subtree, the purpose of the sibling salt in the password subtree is to prevent a guessing attack by the verifier when the subtree has been pruned. In the password subtree, the sibling salt protects the SHoSP and the password from which the SHoSP is derived. However, the password is also protected by the secret salt component of the rich credential. Therefore the role of the sibling salt is less important in the password tree; it is only included as a matter of defense-in-depth: the sibling salt of the password subtree is not entirely redundant because it is generated by the issuer rather than the subject's device; if the random or pseudo-random bit generator used by the subject's device is compromised, the salt generated by the issuer still provides protection against guessing attacks by the verifier.

6.7 Revocable biometric subtrees

6.7.1 Revocable biometrics

Biometric identification entails privacy risks that can be mitigated by the use of privacy preserving biometric techniques. Such techniques are known by many different names, including revocable biometrics, cancelable biometrics, biometric key generation, biometric cryptosystems, biometric encryption, and biometric template protection. A survey of such techniques can be found in [10]. A brief introduction can be found in a Pomcor blog post [11].

In ordinary, non-revocable, one-to-one biometric verification, a verification sample is matched against a biometric template derived earlier from an enrollment sample. Usually, both the enrollment sample and the verification sample are processed to extract features to be used in the comparison. The result of processing a sample is called a feature vector or a biometric code. The biometric template may be the biometric code derived from the enrollment sample, or may result from further processing of the enrollment biometric code to facilitate matching. In some modalities the verification sample may be directly compared to the enrollment sample with no prior feature extraction; in that case the biometric template is simply the enrollment sample.

In revocable biometrics, a biometric helper datum is used instead of a biometric template. A biometric code derived from a biometric sample submitted by the subject is combined with the helper datum to produce a biometric key, in such a way that different but genuine samples (i.e. samples submitted by the legitimate subject rather than by an impostor) consistently produce the same key. The helper datum is such that it is deemed infeasible to derive from it any useful biometric data. Furthermore, the helper datum and the biometric key are randomized, which makes it possible to revoke and replace them if compromised.

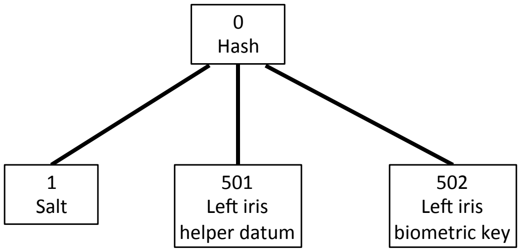

Figure 8: A revocable biometric subtree

One particular methodology for implementing revocable biometrics makes use of an error correction system. The biometric key is randomly generated, a codeword of the error correction system is obtained by adding redundancy to the biometric key, and the helper datum is obtained by x-oring the codeword with an enrollment biometric code derived from an enrollment sample. When a verification biometric code derived from a verification sample is submitted later, it is xored with the helper datum, producing a bit string that differs from the codeword only at those bit positions where the verification biometric code differs from the enrollment biometric code. If the verification sample is genuine, the error correction algorithm is able to correct those bit differences to recover the codeword, and remove the redundancy from the codeword to obtain the biometric key.

The error-correction based methodology was used in [12] for iris recognition.

A rich credential supports the use of both revocable and non-revocable biometric modalities as verification factors.

6.7.2 Revocable biometrics in a rich credential

Figure 8 shows an example of a *revocable biometric subtree* that supports the use of a revocable biometric modality, in the issuance state. It has a peripheral node and three leaf nodes: a salt node, a helper-datum node, and a biometric-key node. The salt node has the undistinguished type 1, and its label is a sibling random salt generated by the issuer.

In the example, the helper-datum node and the biometric-key node have types 501 and 502 and are labeled by a left iris helper datum and biometric key respectively. The issuer obtains an image of the left iris of the subject taken with an infrared camera and uses the image to generate the randomized helper datum and biometric keys. The issuer places the helper datum and the biometric key as the labels of the helper-datum node and the biometric-key node, and uses them to compute the label of the peripheral node, the root label of the tree, and the signature.

The issuer includes the sibling salt and the helper datum in the label array that it delivers to the subject and becomes a component of the rich certificate in the storage state. It also includes the label of the peripheral node, except as discussed below in Section 6.9 and specified in section 6.10.

A revocable biometric subtree supports a biometric verification factor, which may be selected or omitted in a particular presentation of the rich credential to a verifier.

If the biometric factor is omitted, the revocable biometric subtree is pruned from the typed hash tree in the presentation state, by removing the entries for the leaf nodes from the node array, and adjusting the indices of the entries in the label array accordingly. The peripheral node becomes a dangling node.

If the biometric factor is selected for presentation, the subject's device removes the label of the peripheral node from the label array in the transition from the storage state to the presentation state; this preempts a possible bug in the verifier code, which might otherwise fail to verify that the label of the peripheral node is the hash of the types and labels of its children. After receiving the certificate in its presentation state, the verifier obtains a biometric sample from the subject, combines it with the helper datum to compute the biometric key, and adds the biometric key to the label array in the transition from the presentation state to the verification state of the rich certificate.

Use of the sibling salt is a redundant precaution aimed at preventing any leakage of information to the verifier when the biometric factor is omitted.

6.8 Non-revocable biometric subtrees

Figure 9 shows an example of a *non-revocable biometric subtree* that supports the use of a non-revocable biometric modality, in the issuance state. It has a peripheral node and two leaf nodes: a salt node and a biometric-template node. The salt node has the undistinguished type 1, and its label is a sibling random salt generated by the issuer. In the example, the biometric-template node has type 601 and is labeled by a facial image template. The facial image template could be a vector of facial features extracted by the issuer from a

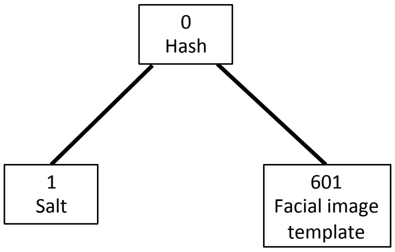

Figure 9: A non-revocable biometric subtree

facial image of the subject, or simply the facial image itself.

The issuer includes the sibling salt and the biometric template in the label array that it delivers to the subject and becomes a component of the rich certificate in the storage state.

A non-revocable biometric subtree supports a biometric verification factor that may be selected or omitted in a particular presentation of the rich credential to a verifier. If the biometric factor is omitted, the subject's device computes the label of the peripheral node and adds it to the label array before pruning the subtree from by removing the entries for the leaf nodes from the node array, and adjusting the indices of the entries in the label array accordingly. The peripheral node becomes a dangling node. If the biometric factor is selected for presentation, the subtree is not modified in the transition from the storage state to the transition state.

The sibling salt prevents the verifier from mounting a guessing attack against the template when the biometric factor is omitted.

6.9 Joint protection of the password and a biometric key against physical capture

The labels of most internal nodes are omitted from the label array in the storage and presentation states to save storage space and communication bandwidth, because they can be recomputed. However the label of the password node is not included in the storage state, hence the root label of the password subtree cannot be recomputed from the labels of leaf nodes when the certificate is transitioned from the storage state to the presentation state. This is a problem if the password subtree is pruned in the transition to the presentation state, in which case the root node of the password subtree becomes a dangling node, and the subject's device has no way of computing its label, short of asking the user for the password even though knowledge of the password is not one of the selected verification factors. This problem can be solved by keeping the root label of the password subtree in the storage state.

The same problem occurs in connection with any revocable biometric modalities supported by the rich credential because the biometric keys are omitted from the storage state, and the same solution is available.

However, if the rich credential includes a revocable biometric modality and it is known at issuance time that both the the password and the revocable biometric will be used as verification factors in every presentation of the credential, it is best to not keep the root labels of the password subtree and the revocable biometric subtree in the storage state. Omitting those two labels prevents an adversary who physically captures the credential from mounting separate guessing attacks against the password and the biometric key using the root labels of the subtrees. The adversary can only mount a joint guessing attack against the password and the biometric key by testing guesses against the signature in the certificate, and that attack will fail if the combined entropy of the password and the biometric key is sufficiently high.

6.10 Rich credential issuance protocol

The issuance protocol for a rich credential has two stages. Stage 1 takes place only once. Stage 2 may be repeated for each computing device where the subject wants to install a credential, as described below in Section 6.13.

6.10.1 Stage 1

In Stage 1, the issuer creates a record for the subject containing attributes and biometric verification data, retrievable by a security code. The subject's record contains biometric templates for the non-revocable biometric modalities only. Verification data for revocable biometric modalities is obtained in Stage 2 and is not stored in the subject's record, for reasons discussed in Section 6.13.

Stage 1 may involve in-person and/or remote interactions between the subject and the issuer. Remote interactions may include one or more remote-identity proofing events with credentials provided by upstream identity sources.

After the subject's record has been assembled, the issuer provides the security code to the subject. For stronger security, the code may be split into two or more portions delivered through different channels. For example, if Stage 1 takes place during an in-person visit of the subject to the issuer, the issuer may provide a portion to the subject in the course of the visit and send another portion to the subject's address of record.

6.10.2 Stage 2

Stage 2 takes place remotely between the subject's device and the issuer. It comprises the following steps:

1. The subject's device generates a key pair for a public key cryptosystem and a random high entropy bit string to be used as the secret salt.

2. The subject's device prompts the subject for the password and computes the hash of the password and the secret salt, i.e. the SHoSP.

3. If the credential is to include a revocable biometric modality, the subject's device obtains a biometric sample for the modality (e.g. an iris image). This step is repeated if the credential is to include multiple revocable modalities.

4. The subject's device establishes a TLS connection (or some other secure connection) to the issuer with authentication of the issuer during the handshake, and sends the following data over the connection:

   - The security code.
   - The public key component of the key pair.
   - The SHoSP.
   - Biometric samples for revocable biometric modalities to be included in the credential, if any.

5. The subject's device proves knowledge of the private key component of the key pair to the issuer over the TLS connection in a manner dependent on the kind of public key cryptosystem of the key pair, as described below in Section 6.12.

6. The issuer generates the randomized helper datum and biometric key for each revocable biometric data from the samples received from the subject's device.

7. The issuer constructs the node array of the typed hash tree of the rich credential, with attribute subtrees for the attributes in the subject's record, a password subtree, a non-revocable biometric subtree for each template in the subject's record (if any), and a revocable biometric subtree for each revocable biometric sample received from the subject's device. The issuer may also include entries for dangling nodes in the node array, to hide the fact that certain subtrees have been omitted, e.g. because they are not applicable to the subject of the rich credential.

8. The issuer fills in the labels of the leaf nodes in the label array, which include:

   - The sibling salts, which it generates using a random or pseudo-random bit generator.
   - The attribute values found in the subject's record (if any).
   - The SHoSP received from the subject's device.
   - The biometric templates found in the subject's record.
   - The randomized helper datum and biometric key for each revocable biometric modality (if any) generated from the samples received from the subject's device.

9. The issuer computes the root label of the typed hash tree, filling in all the entries for the internal nodes in the label array in the process.

10. The issuer assembles the metadata of the rich certificate, and computes the signature on the public key, the metadata and the root label of the typed hash tree.

11. The issuer *removes* the following labels from the label array, to put the array in the storage state before transmitting it to the subject's device:

    - The labels of the central nodes. (Recall that the central nodes are the internal nodes other than the roots of the peripheral subtrees.)
    - The root labels of the attribute subtrees.
    - The root labels of the non-revocable biometric subtrees.
    - The SHoSP that labels the password node.
    - The biometric keys that label nodes of the revocable biometric subtrees, if any.

12. As discussed above in Section 6.9, if the rich credential includes a revocable biometric modality and it is known that both the password and the revocable biometric will be used as verification factors in every presentation of the credential, the issuer also removes the root labels of the password subtree and the modality.

13. The issuer assembles the rich certificate, comprising the public key, the metadata, the node and label arrays, and the signature, and sends it to the subject's device over the TLS connection.

14. If the public key of the issuer is not generally known, the issuer also sends to the subject's device over the TLS connection a certificate chain comprising the issuer's certificate and zero or more CA certificates.

15. The subject's device assembles the rich credential, comprising the private key, the secret salt and the rich certificate and stores it, together with the issuer's certificate chain, if received from the issuer. The rich certificate is received from the issuer in the storage state, and is stored without modification.

6.11 Rich credential presentation and verification protocol

The presentation and verification protocol comprises the following steps:

1. The subject's device establishes a TLS connection (or some other secure connection) to the verifier with authentication of the verifier during the handshake. All protocol communications between the subject's device and the verifier take place over the TLS connection, which is resumed or reestablished as needed if lost during the protocol.

2. After the TLS connection has been established, the subject and the verifier negotiate the attributes to be disclosed, the biometric modalities to the presented as verification factors, and whether knowledge of the password is to be included as a verification factor.

   For example, a simple negotiation process could be as follows: the verifier specifies the attributes and verification factors that it requires; the subjects's device determines if corresponding attribute and verification factor subtrees are present in the typed hash tree and cancels the presentation if they are not; the subject's device asks the subject for consent to disclose the requested attributes and present the requested verification factors and cancels the presentation if consent is not granted; the negotiated attributes and factors are those requested by the verifier if the subject's device has not canceled the presentation.

3. If knowledge of the password has been negotiated to be included as a verification factor, the subject's device prompts the subject for the password and computes the SHoSP.

4. The subject's device transitions a copy of the rich certificate from the storage to the presentation state as explained above in sections 6.5–6.8, according to the result of the negotiation, and sends the presentation state copy to the verifier, together with the SHoSP if knowledge of the password has been negotiated to be included as a verification factor. (Equivalently, the SHoSP may be included in the presentation state copy as the label of the password node instead of being sent separately.)

5. The subject's device proves knowledge of the private key component of the key pair to the verifier over the TLS connection in a manner dependent of the kind of public key cryptosystem of the key pair, as described below in Section 6.12.

6. For each revocable biometric modality whose inclusion in the presentation has been negotiated:

   - The subject's device arranges for transmission of a modality sample from the subject to the verifier in a manner that allows the verifier to perform presentation attack (spoofing) detection.
   - The verifier verifies that no presentation attack is taking place and computes the biometric key from the helper datum found in the label array and the transmitted sample.

7. The verifier transitions the rich certificate from the presentation state to the verification state by:

   - Adding the SHoSP to the label array, if knowledge of the password is a negotiated verification factor and the SHoSP was sent separately at step 4.

- Adding to the label array the biometric key of any revocable biometric modality that has been negotiated to be included as a verification factor.

8. The verifier computes the root label of the typed hash tree from the labels of the leaf nodes (including the labels of any dangling nodes resulting from the pruning of subtrees or originally present in the issuance state).

9. The verifier validates the rich certificate, checking that it has not expired or been revoked, verifying the signature on the public key, the metadata and the root label of the typed hash tree, and validating the issuer's certificate chain, if received from the subject's device.

10. (This step is postponed until this point to save the subject the trouble of providing a sample for a non-revocable biometric modality if something else goes wrong, e.g. if the subject mistypes the password. Providing a sample for a revocable modality cannot be similarly postponed because the biometric key is used in the computation of the root label of the hash tree and the certificate signature.) For each non-revocable biometric modality whose inclusion in the presentation has been negotiated, the subject's device arranges for transmission of a biometric sample from the subject to the verifier in a manner that allows the verifier to perform presentation attack detection. The verifier verifies that no presentation attack is taking place and the transmitted sample matches the biometric template for the modality found in the label array.

11. The verifier examines the attributes asserted by the issuer in the typed hash tree (represented by key-value pairs with keys in the node array and values in the label array) and relies on them to authorize a transaction or register the subject as a subscriber to a service.

6.12 Proving knowledge of a private key over a secure connection in different kinds of cryptosystems

A rich credential includes a key pair pertaining to a public key cryptosystem. The subject's device proves knowledge of the private key component of the key pair to the issuer at step 5 of the issuance protocol, and to the verifier at step 5 of the presentation and verification protocol. How this is done depends on the public key cryptosystem that is used, which one is used being determined by the rich credential specification referenced by the version number field in the rich certificate.

In the following protocol descriptions we shall refer to the subject's device as the *prover*, and to the party that verifies possession of the private key as the *verifier*, whether that party is playing the role of credential verifier or credential issuer. All interactions take place over a previously established secure connection (e.g. a TLS connection) between the prover and the verifier, after the verifier has demonstrated its identity (e.g. a domain name included in a TLS server certificate) to the prover and received the prover's public key.

The protocols use the verifier's identity demonstrated during connection establishment to prevent a man-in-the-middle attack where the attacker impersonates the prover by relaying protocol messages between the prover and the verifier. Such a man-in-the-middle attack against authentication is known as a *mafia attack* [13] or a *chess grand master attack* [14].

The following protocol is used if the public key cryptosystem is a digital signature cryptosystem, such as ECDSA, DSA, or RSA:

1. The verifier sends a nonce to the prover.

2. The prover generates a nonce and computes a joint hash of both nonces and the verifier's identity.

3. The prover signs the joint hash with the private key.

4. The prover sends its nonce and the signature to the verifier.

5. The verifier computes the joint hash of both nonces and its own identity.

6. The verifier verifies the signature on the joint hash using the public key

The following protocol is used if the public key cryptosystem is an encryption cryptosystem, such as RSA or El Gamal Encryption:

1. The verifier generates a nonce and sends the prover a message encrypted with the prover's public key containing the nonce and a joint hash of the nonce and the verifier's identity.

2. The prover decrypts the message and verifies the hash. If hash verification fails the prover does not complete the protocol. Otherwise it sends the nonce to the verifier.

3. The verifier verifies that the nonce is equal to the one it sent.

The following protocol is used if the public key cryptosystem is a key exchange cryptosystem such as Diffie-Hellman (DH) or Elliptic-Curve Diffie-Hellman (ECDH).

1. The verifier generates a nonce and an ephemeral DH key pair, derives a symmetric encryption key from the ephemeral private key and the prover's public key, and sends the prover the ephemeral public key and a message encrypted with the symmetric key, containing the nonce and a joint hash of the nonce and the verifier's identity.

2. The prover validates the public key received from the verifier. Public key validation routines can be found in [15, 5.6.2.3.1] for DH and [15, 5.6.2.3.2] for ECDH. If validation succeeds, the prover derives the symmetric key, decrypts the message and verifies the hash. If public validation or hash verification fails, the prover stops. Otherwise the prover sends the nonce to the verifier.

3. The verifier verifies that the nonce is equal to the one it sent.

6.13 Installation of rich credentials in multiple devices

Stage 2 of the issuance protocol can be run multiple times to install credentials in multiple devices. However, the security code that the subject uses to request a credential should be a one-time code, or, if split into one or more portions, one of the portions should be a one-time code, to prevent an adversary from obtaining a credential just by capturing a security code kept in long term storage.

After using the one-time code to install a credential in a device, the subject must therefore be able to obtain subsequent one-time codes to install credentials in subsequent devices. If the subject has access to a rich credential in one device, he or she can obtain a one-time code for another device by requesting it remotely from the issuer, using the rich credential for authentication. Otherwise, if the subject has a reusable portion of the original security code, he or she may be allowed to obtain a one-time portion to be used in combination with the reusable portion by remotely presenting a sample of a non-revocable biometric modality to be matched against a biometric template in the subject's record kept by the issuer, with presentation attack detection, and requesting the one-time portion to be sent by physical mail, telephone, text messaging or email to a destination known to be controlled by the subject. Otherwise the subject should be required to repeat Stage 1.

Credentials installed in different devices are constructed with the same attributes, and the same biometric templates of non-revocable modalities, but different key pairs and different helper data items of revocable biometric modalities. They may be constructed using the same or different passwords, at the subject's discretion.

The reason for using different helper data items in different devices is that a helper data item should not be stored in the subject's record together with the corresponding biometric key, because useful biometric information could be derived by an adversary who captures both. And storing the helper data item by itself would not obviate the need to ask the subject for a biometric sample in Stage 2. It would be more convenient for the user to simply store a biometric sample in the subject's record, but then it would not be possible to claim that no useful biometric information pertaining to the revocable biometric modality is ever kept in persistent storage.

7 Solution 1: rich credential issued by a DMV

In Solution 1, we propose the use of a rich credential issued by a DMV as the primary evidence in a remote identity proofing event. Relying on a DMV for a remote identity proofing solution is a natural choice, because a driver's license or non-driver identity card is the default piece of evidence used for in-person proofing in the US. On the other hand, a problem with this choice is that a majority of the more than 50 DMVs in the US must be willing to support the solution in order for the solution to be deployed at population state. To mitigate this problem, we have designed Solution 1 so that it is very easy and inexpensive to implement by a DMV.

A DMV already collects a facial image from each of its customers for use in a driver's license or non-driver identity card. Therefore we propose the use of facial recognition as a non-revocable biometric modality in the rich credential. For the sake of simplicity, no other modality is used, and the facial biometric template is the unprocessed facial image itself, all image processing being done by the verifier. Biometric verification includes presentation attack detection by submission of an audio-visual stream of the subject reading prompted state, but only the verifier, not the DMV, is concerned with this.

Since a rich credential supports selective presentation of verification factors, the verifier may specify whether it requires facial recognition or not. It may also specify whether it requires knowledge of the password as a verification factor, i.e. submission by the subject's device of the salted hash of the password and the secret salt. A rich credential also supports selective disclosure of attributes, so the verifier can also request specific attributes. The user is asked for consent to present the required verification factors and disclose the requested attributes.

To facilitate wide deployment, the rich credential is kept by the subject in JavaScript persistent storage provided by a web browser, often referred to as *HTML5 local storage* [16], rather than in a smart card or a dongle. The credential is placed in local storage by the JavaScript front-end of a credential-issuance web application of the DMV. A request for presentation of the credential to a verifier is intercepted by a *service worker* [17] that the DMV front-end registers with the browser, and is fulfilled by JavaScript code generated by the service worker, without any involvement from the back-end of the DMV web application. The fact that the back-end is not involved in credential presentation greatly reduces the computational demands on the DMV data center. It also means that the DMV cannot tell how the subject uses the credential without colluding with the relying parties, an important privacy feature known as *unobservability* [8]. Since the DMV JavaScript front-end runs on the browser, the fact that the DMV does not observe the presentations of the credential is auditable by the subject.

Figure 10 is a swimlane diagram of the issuance of a rich credential by a DMV and its presentation to a verifier. Issuance and presentation of a rich credential were described in general terms in sections 6.10 and 6.11. Here we describe the specifics that apply to a DMV rich credential. All communications take place over TLS connections, where the DMV and the verifier play the role of TLS servers and authenticate with TLS certificates.

In Stage 1 of the issuance process, the DMV determines the subject's attributes, obtains the facial image, and stores the attributes and image in a record for the subject together with a security code that it provides to the subject and the subject uses in Stage 2 to retrieve the credential.

The security code may be split into two portions that are provided to the subject through different channels. Usually, Stage 1 will take place during

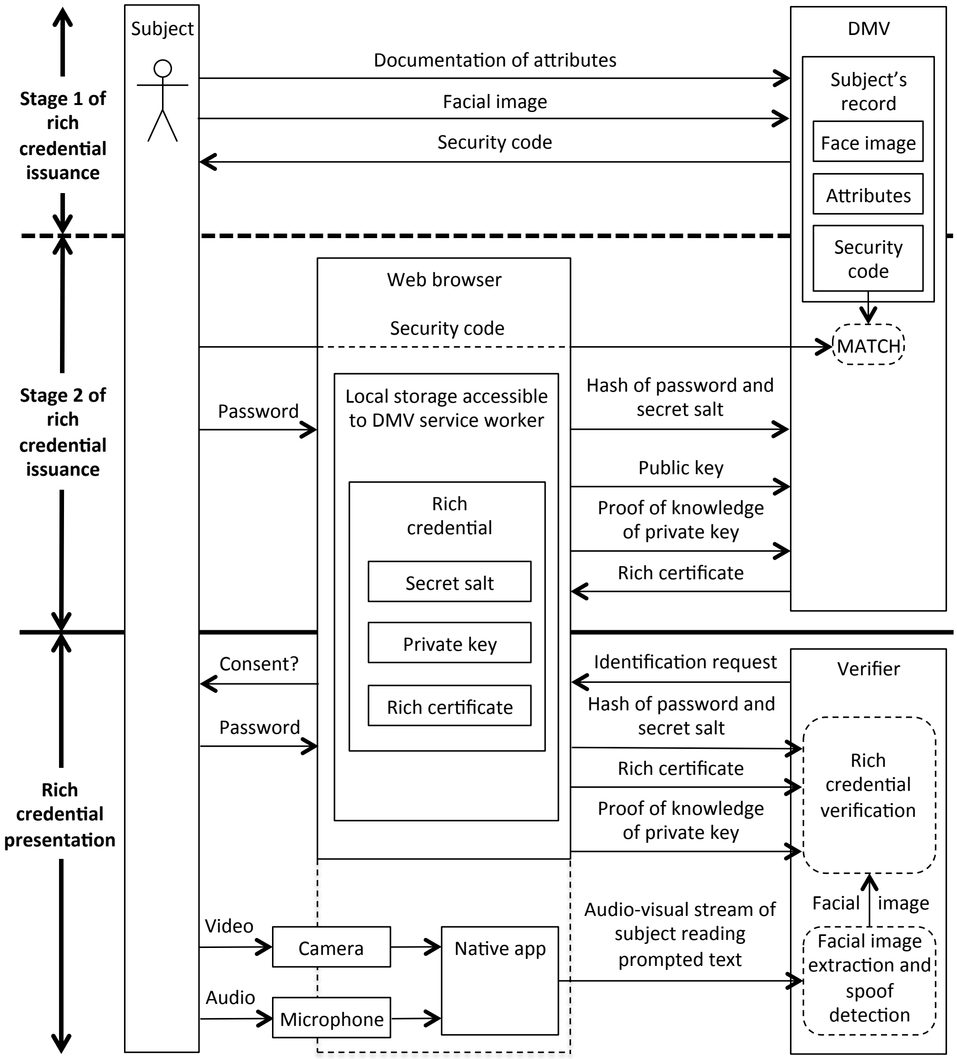
Stage 1 of rich credential issuance
Subject
Documentation of attributes
Facial image
Security code
DMV
Subject's record
Face image
Attributes
Security code
Stage 2 of rich credential issuance
Web browser
Security code
MATCH
Password
Local storage accessible to DMV service worker
Hash of password and secret salt
Public key
Rich credential
Proof of knowledge of private key
Secret salt
Rich certificate
Private key
Rich certificate
Consent?
Identification request
Verifier
Hash of password and secret salt
Password
Rich certificate
Rich credential verification
Rich credential presentation
Proof of knowledge of private key
Facial image
Audio-visual stream of subject reading prompted text
Video
Camera
Native app
Facial image extraction and spoof detection
Audio
Microphone

Figure 10: Issuance and presentation of a DMV rich credential

an in-person visit of the subject to the DMV. In that case one portion of the security code may be given to the subject in person, while the other may be sent to the address of record.

Stage 1 may also take place remotely, if the subject is able to provide sufficient identity evidence using credentials previously issued by upstream identity sources. For example, if the subject moves to a different state, he or she may be able to obtain both a rich credential and a physical driver's license from the new state of residence without a visit to the DMV by presenting a rich credential from the old state along with evidence that the subject has moved to a a new address in the new state. In that case, one portion of the security code may be provided to the subject online as part of the remote identity proofing event where he or she presents the rich credential from the old state, while the other portion is sent through the post to the new address.

Stage 2 of the issuance process takes place remotely and may be repeated by the subject to install credentials in several devices, as discussed above in Section 6.13. The subject initiates Stage 2 by accessing the credential issuance web application of the DMV and sending the security code (both portions if split), which the DMV application uses to locate the subject's record.

After locating the record, the DMV application responds with an HTML page containing front-end JavaScript code. The front-end code generates a key pair and a secret salt, asks the subject for a password, computes the hash of the password and the secret salt, sends the public key and the salted hash to the back-end of the DMV application, and proves knowledge of the private key to the back-end of the DMV application. The back-end constructs the rich certificate as described in Section 6.10 and downloads it to the front-end. The front-end assembles the credential, stores it in HTML5 local storage, and registers a service worker with the browser, which will take care of presenting the credential to verifiers.

Presentation and verification of the rich credential is initiated by a redirection from a verifier web application to the DMV web application, which is intercepted by the DMV service worker. The redirection conveys an identification request, which specifies the attributes that the verifier wants to see, and whether the verifier requires face recognition or submission of the salted hash of the password and the secret salt, or both, or neither.

Without involving the back-end of the DMV application, the service worker constructs and displays a web page that describes the identification request and the verifier web application making the request and asks the user for consent. If the verifier requests submission of the salted hash of the password, the consent page contains an input field for the subject to enter the password.

If the subject grants consent, the JavaScript credential-presentation code in the web page retrieves the rich credential from the HTML5 local storage. If the salted hash of the password is to be submitted (as assumed in the figure), the credential-presentation code computes it from the password entered by the subject and the secret salt in the credential, and sends it to the verifier. The credential-presentation code transitions a copy of the rich certificate to the presentation state, and sends it to the verifier along with the issuer's certificate chain if included in the credential. As described in more detail in Section 6.11, transitioning the rich certificate to the presentation state involves pruning peripheral subtrees from the typed hash tree of the credential. The pruned subtrees include the attribute subtrees for the attributes not to be disclosed, the password subtree if the salted hash of the password is not to be submitted, and the subtree for the facial image biometric modality if face recognition is not to be used. The credential-presentation code also proves knowledge of the private key to the verifier by one of the methods described above in Section 6.12.

After receiving the certificate and the salted hash of the password, and verifying the proof of knowledge of the private key, the verifier transitions the certificate to the verification state by adding the salted hash of the password to the label array. Then it computes the root label of the typed hash tree, verifies the signature and validates the certificate.

If verification and validation succeed, and if face recognition is required (as assumed in the figure), the verifier launches a native application on the same device where the browser is running or on a different device, which streams an audio-visual stream of the subject upon which the verifier performs face recognition with presentation attack (spoofing) detection.

If the subject's device is a traditional laptop or desktop and the native application runs on that device, the verifier's web application can launch the native application using an HTTP response with a Content-Type header that the native application has been registered to handle. If the subject's device is a mobile device and the native application runs on that device, the verifier's web application can launch it by redirecting the browser to a custom URL scheme handled by the native application. If the native application runs on an ancillary mobile device other than the device where the browser is running, the verifier provides the subject with a security code that identifies the credential presentation and asks the subject to manually launch the native application, direct it to the verifier, and enter the security code into the native application for authentication to the verifier.

The native application transmits to the verifier an audio-visual stream of the subject reading prompted text. The text is randomly selected or generated by the verifier with high entropy, to ensure that an adversary cannot replay an audio-visual recording of the subject reading the same text. The verifier uses face recognition software to match the facial image in the label array to the face shown in the video channel of the stream, and speech recognition to verify that the text being read is the prompted one.

The verifier must also verify that the audio and video channels are in synchrony, to prevent an adversary from combining a recorded video of the subject with live audio of the adversary reading the prompted text into an MP4 or other container stream.

To verify synchrony, the verifier checks that the lip movement in the video channel of the stream is synchronized with the utterances in the audio channel. It has been shown in [18] that it is possible to automatically analyze lip movement in a facial video to the extent of lip reading a sequence of digits. The prompted text is arbitrary text rather than a sequence of digits, and lip reading is difficult even for humans, because there are about three times as many phonemes than visemes in English, and because, while some visemes are easy to identify, others are obscured by coarticulation [19, 20]. However all that the verifier needs to do is to correlate the timing of *easily distinguishable visemes* with corresponding phonemes.

If the subject's face is recognized and no presentation attack is detected, the presentation of the rich credential has been successful and the verifier may then authorize a transaction or register the subject as a subscriber to a service.

8 Security analysis

8.1 General threat model for all five solutions

8.1.1 Adversarial goals

Remote identity proofing adversaries may have the following goals:

Impersonation An adversary may want to impersonate a subject vis-à-vis a verifier, by capturing evidence pertaining to the subject, falsifying evidence, or tampering with the protocol for remote presentation of the evidence.

Fraudulent claiming of attributes A subject who has obtained identity evidence may want to claim attributes that he or she is not entitled to by modifying the evidence or tampering with the protocol for presenting the attributes to a verifier.

Repudiation A subject who has presented evidence in an identity proofing event in order to perform a transaction or register for a service may want to claim later that he or she has not participated in the event and performed the transaction or registered for the service.

False implication A verifier may want to implicate a subject in a remote identity proofing event that has not taken place or has involved a different subject.

PII capture An adversary may want to capture personally identifiable information (PII) pertaining to a subject that is included in a remote identity proofing credential or exposed in the process of acquiring or presenting the credential.

8.1.2 Adversaries

Remote identity proofing procedures may face the following categories of adversaries:

Subject A subject who has obtained identity evidence is a potential adversary who may want to claim attributes not asserted by the identity source, impersonate a different subject, or repudiate his or her participation in a proofing event.

Identity source insider An identity source has to be trusted and therefore cannot be counted as an adversary. But insiders with limited operational roles at the identity source may not be fully trusted and may be counted as adversaries.

Verifier or verifier insider The subject must disclose some verification data to the verifier, but the verifier or its personnel may be counted as potential adversaries if they try to obtain subject data not meant to be disclosed during a proofing event, or use subject data to impersonate the subject vis-à-vis a different verifier.

Network attacker Remote identity proofing takes place across the Internet and is exposed to attack over the network by any party with access to the Internet. A network attacker may also be able to extract identity evidence or PII from a computing device that carries a remote identity proofing credential by interacting with the device or planting malware on the device.

We generally assume that a network adversary is not able to breach the security of a TLS connection established from a subject-controlled computing device to either an identity source or a verifier with destination authentication, nor to impersonate the destination.[1]

(In the papers on solutions 3-5, which use NFC technology, we will discuss the capabilities of adversaries within NFC attack range of a device carrying identity evidence.)

Physical attacker A physical attacker is an adversary who tries to capture a device containing digital data, or capture or copy a document carrying printed data, the latter being relevant to Solution 5.

Identity sources and verifiers may use virtual servers and databases hosted in computing clouds. The cloud operator and other cloud tenants are potential adversaries. However, for simplicity, we shall not consider a separate category of cloud attackers. If an identity source uses a computing cloud, we shall include the operator and other tenants of that cloud in the category of *identity source insiders*. Similarly, if a verifier uses a computing cloud, we shall consider the operator and other tenants of that cloud in the category of *verifier or verifier insiders*.

8.2 Adversarial capabilities specific to Solution 1

Solution 1 faces three particular kinds of identity source insider adversaries:

[1] Unfortunately, this assumption often fails to be satisfied in practice, because TLS is repeatedly affected by protocol or implementation vulnerabilities. However, finding mitigations for the lack of secure connection capability would be difficult, and implementing such mitigations is likely to be impractical. The best mitigation against TLS security woes would be to define and deploy a better protocol [21, 22].

1. A developer working on the DMV web application, either in-house or at a contractor or software vendor, may be able to tamper with the code, causing it to plant malicious JavaScript in the web application front-end. That malicious code will run in the subject's browser with the same origin as the DMV web application, and may therefore be able to capture the rich credential, including its private key component, and send it to the adversary.

2. A software engineer at the DMV or at a contractor working for the DMV or a vendor supplying software to the DMV may be able to tamper with the pseudo-random bit generator (PRBG) that is used to generate the random salts used in rich credentials and the labels of any dangling nodes that may be present in the issuance state of the rich certificate. While the key pair used to sign credentials is likely to be generated in a hardware security module (HSM) using a secure PRBG, the random salts may not be generated inside an HSM and little attention may be paid to the security of the PRBG used to generated them.

3. Personnel who interact with the subject during the in-person phase of credential issuance are in charge of providing the security code to the subject, and are therefore well placed to copy it and use it to retrieve the credential issued to the subject.

A network attacker may be able to exploit a cross-site scripting (XSS) vulnerability in the DMV web application to plant malicious JavaScript code in the subject's web browser that will run with the same origin as the DMV web application, and may then be able to capture the rich credential, including its private key component, and send it to the adversary.

A network attacker may also be able to capture the rich credential using malware that takes control of the subject's device, and may also capture the subject's password and biometric samples as they are entered.

A physical attacker may try to capture the subject's computing device with the intention of extracting the rich credential from browser storage.

A physical attacker may also try to steal or copy the document containing the security code, given to the subject during the in-person visit to the DMV. Such physical attacker may be a DMV insider as noted above, or some other adversary.

8.3 Threats and mitigations pertaining to Solution 1

Threat 1: malicious JavaScript code

An adversary plants malicious JavaScript code in a web page returned by the DMV web application to the subject's browser. That code runs in the browser with same origin as the DMV application and is able to extract the rich credential, including the private key, the secret salt and the rich certificate, from HTML5 local storage. The malicious JavaScript code may be planted by one of the following means:

- It may be planted in a JavaScript library used by the DMV web application.
- It may be placed by a cross-site scripting (XSS) attack in a string included in the HTML source of the web page, which the application fails to sanitize.
- It may be planted by a malicious insider at the DMV in the front-end of the DMV web application.

Malicious JavaScript code may also be carried by advertisements, but no advertisements are presumably placed in the web site of a DMV.

It should be noted that capturing the private key is not sufficient to impersonate the subject vis-à-vis a verifier that requires presentation of all three verification factors supported by the rich credential.

Mitigations of Threat 1

Mitigation 1a If the experimental W3C Web Cryptography API [23] is used to generate the key pair, the resulting public and private keys are contained in CryptoKey objects, and the private key can be made non-extractable from its CryptoKey object. The CryptoKey object is not persistent and cannot be converted to a string for storage in HTML5 local storage, but it may be stored in an alternative persistent browser storage made available through the IndexedDB API [24].

Making the private key non-extractable does not protect it from being captured by the code that generates the key pair, nor from used *in situ* by malicious JavaScript code with same web origin as the DMV application, but protects it against being extracted from the subject's device and used elsewhere.

Mitigation 1b The DMV web application that issues rich credentials and registers the service worker responsible for the presentation of rich credentials uses a DNS domain different from the DNS domain of the DMV web site (both may be subdomains of a common DNS domain registered by the DMV). Developers of the web application are trained to avoid XSS vulnerabilities and the use of unnecessary JavaScript libraries. Any libraries that need to be used are carefully inspected before deployment to ensure that they don't contain malware, and the web application code is carefully reviewed and screened for XSS vulnerabilities. This, however does not prevent an attack by an identity source insider.

Mitigation 1c The tasks of generating the key pair, storing the rich certificate and private key in HTML5 local storage, asking the user for consent to present the credential to a verifier, and presenting the credential, are carried out by a service worker, but the service worker is registered by a fourth party, independent of the DMV that issues the credential and of the verifiers, which acts on behalf of the subject and is freely chosen and trusted by the subject. We shall refer to such a service worker as a *consent manager*. The consent manager mitigates Threat 1 because only JavaScript code with same origin as the consent manager has access to the rich credential.

Threat 2

An adversary plants malware in the subject's computing device that hosts the browser where the rich credential is stored, or in the device that captures the audio-visual stream used by the subject to present a facial image to the verifier, if different. Such malware may be able to capture any of the following:

- The rich credential kept in the browser's HTML5 local storage including the private key, the secret salt, the attributes, and the biometric verification data.
- The password when entered by the user, by means of a key logger.
- The audio-visual stream.

Partial mitigation of Threat 2

The private key can be protected by storing the rich credential in a cryptographic module and making the private key not extractable. The cryptographic module may be located in a smart card or dongle connected to the computing device that hosts the web browser; this requires browser support or a browser plug-in. The cryptographic module may also be include in a Trusted Execution Environment (TEE), a Secure Element, or a Trusted Platform Module (TPM) within the computing device; this requires support by the platform provider.

Threat 3

An adversary physically captures the subject's computing device that hosts the browser where the rich credential is stored, and reads the rich credential from persistent storage.

Mitigations of Threat 3

Mitigation 3a Modern computing devices provide protection against this threat by encrypting all user data, including browser storage, under a key or key hierarchy derived from a password and a device key that is stored in or built into tamper resistant storage. The subject can mitigate Threat 3 by using a device that provides such protection and activating the protection when the device is not being actively used.

Mitigation 3b The rich credential can be protected against physical capture by storing it in a tamper-resistant hardware such as a smart card, a Hardware Security Module (HSM) in a dongle form factor, a Secure Element, or a TPM.

Threat 4

A physical attacker, which may or may not be a DMV insider, captures the security code provided to the subject for retrieval of the rich credential.

Mitigation of Threat 4

The security code needed to retrieve the credential is split into two portions, a *reusable portion* and a *one-time portion*, as already mentioned above. The reusable portion is provided to the subject in Stage 1 of the credential issuance process, typically during the in-person visit to the DMV, possibly as the result of a remote identity proofing event involving an upstream identity source. The one-time portion is sent by mail to the address of record of the subject, by personnel belonging to a separate organization within the DMV. The reusable portion is kept by the subject in secure storage and can be reused as needed to retrieve credentials for use in computing devices owned by the subject, or for replacement of a lost credential, as described above in Section 6.13.

Threat 5

A network attacker breaches the security of the DMV data center to capture the record containing the subject's attributes and facial image.

Mitigation of Threat 5

The subject's record is taken offline after the rich credential is issued to the subject's first device. If the subject subsequently requests a new one-time portion of the security code in order to retrieve a credential for another device, the subject's record is put back online when the new one-time portion is mailed to the subject.

Threat 6

A subject repudiates participation in a remote identity proofing event, or a verifier falsely implicates a subject in an event.

Defense against fraudulent repudiation of a transaction involving a cryptographic credential is usually based on the fact that the private key component of the credential is controlled by the subject, is generated within a subject-controlled device, and never leaves the device. In Solution 1, however, this defense is not available because the private key is stored in HTML5 local storage that the same origin policy of the web makes accessible to the DMV web application.

Mitigation of Threat 6

The usual defense against fraudulent repudiation can be made available by the Mitigation 1c of Threat 1 and the mitigation of Threat 2 described above.

The verifier can also defend against non-repudiation by recording the audiovisual stream presented by the subject and retaining the recording, if the facial recognition verification factor is not omitted.

The subject can defend against false implication by carefully protecting the private key, and promptly reporting a theft or compromise of the device that carries the private key.

8.4 Security posture of Solution 1

The security posture of Solution 1 can be summarized as follows:

- Subtrees can be pruned from the typed hash tree of the rich credential, allowing for selective disclosure of attributes, and selective presentation of verification factors, i.e. omission of password verification and/or face recognition. But none of the following can be done without invalidating the signature in the credential:

  - Adding or modifying attributes.
  - Using a different password than the one whose salted hash was sent to the issuer in Stage 1 of the credential issuance process.
  - Adding a biometric modality to the credential.
  - Modifying the facial image in the credential.

  It should be noted that, although a rich credential provides selective disclosure of attributes like an Idemix Anonymous Credential or a U-Prove Token, it does not provide unlinkability. Even if no identifying attributes are disclosed, credential presentations can be linked to each other and to credential issuance by the signature on the credential and by metadata such as the serial number. The rich credential does provide unobservability of presentations by the issuer, as discussed above in Section 7.

- Assuming the above mitigation of Threat 4, an adversary who tries to retrieve from the DMV the rich credential issued to a subject must capture two portions of a security code: a reusable portion usually provided in person to the subject, and a one-time portion that is mailed to the address of record of the subject.

- If the verifier requires all three verification factors supported by the rich credential, impersonation of the subject requires:
  - Knowledge of the private key.
  - Knowledge of the password.
  - Submission of an audio-visual stream that spoofs the subject reading prompted text randomly selected or generated by the verifier. The verifier verifies the synchrony of the audio and visual channels by tracking lip movement in the video channel and correlating distinguishable visemes to phonemes identified in the audio channel.
- The rich credential is vulnerable to capture by malicious JavaScript code, but only if the malicious code has the same web origin as the credential-issuance web application of the DMV. Mitigations 1a–c are available to address this vulnerability.
- The rich credential is vulnerable to capture by malware planted in the device that hosts the browser. This includes the private key, unless the above partial mitigation of Threat 2 is used, at a high usability cost.
- The subject can protect the private key against physical capture by using a modern computing device that protects user data by encrypting user data, including browser storage, under a key or key hierarchy derived from a password and a device key that is stored in or built into tamper resistant storage, and activating such protection when the device is not being actively used.
- The salted hash of the password is vulnerable to capture by a DMV insider when it is supplied in Stage 1 of the credential issuance process, or by a verifier or verifier insider when it is submitted together with the rich credential. If captured, it can be used in a fraudulent presentation of the rich credential.
- The password is not vulnerable to capture by a DMV insider, nor by a verifier or verifier insider. If reused by the subject at web sites and captured at a web site, it cannot be used in a fraudulent presentation of the rich credential.

- The password is not vulnerable to a security breach at the DMV data center or at a verifier data center, but it is vulnerable to an offline guessing attack by an adversary that captures the rich credential stored in the browser.

- The verifier cannot mount a guessing attack against any data that is not used in a presentation of the rich credential, including omitted attributes, the salted hash of the password if password verification is omitted, and the facial image if facial recognition is omitted.

- The verifier cannot rely on the private key not being shared in a defense against fraudulent repudiation of a presentation of the credential, but can instead rely on a recording of the audiovisual stream presented by the subject.

Solution 1 provides arguably stronger identity assurance than in-person presentation of a physical driver's license, for the following reasons:

1. It provides three verification factors, whereas a physical driver's license provides only two.

2. Driver's licenses can be forged. There is a market for forged driver's licenses, where they sell for $150 each [25]. By contrast, a rich credential issued by a DMV cannot be forged.

9 Conclusion

This paper has described Solution 1, the first of five remote identity proofing solutions that we have identified as possible alternatives to knowledge-based verification. Solution 1 is based on the concept of a *rich credential*, a new kind of cryptographic credential that can be used by a subject to remotely present multiple verification factors to a verifier with whom the subject may have no prior relationship, including something that the user has (a private key), something that the user knows (a password), and something that the user "is" (one or more biometric features).

The concept of a rich credential is based on another new concept, the concept of a *typed hash tree*, a tree that can be used to represent a collection of key-value pairs and whose root label provides an omission-tolerant cryptographic checksum of the collection. The omission tolerance of a typed hash tree is used in a rich credential to provide selective disclosure of attributes and selective presentation of verification attributes.

In Solution 1, the rich credential contains a facial image, which the verifier matches against the subject's face shown in an audio-visual stream of the subject reading prompted text. The verifier verifies the synchrony between the audio and video channel of the stream by tracking the subject's lips in the video stream and matching distinguishable visemes to phonemes in the audio stream.

Solution 1 provides arguably stronger identity assurance than in-person presentation of a physical driver's license because it provides three verification factors rather than two and it cannot be forged.

10 Acknowledgments

This paper is the result of a project sponsored by an SBIR Phase I grant from the US Department of Homeland Security. It does not necessarily reflect the position or the policy of the US Government.

We are very grateful to Prof. Phil Windley for many comments on project working documents.

References

[1] Francisco Corella. Pomcor Receives DHS Grant to Look for Alternatives to Knowledge-Based Verification for Remote Identity Proofing. Pomcor blog post, August 3, 2016. https://pomcor.com/2016/08/03/pomcor-receives-dhs-grant-to-look-for-alternatives-to-knowledge-based-verification-for-remote-identity-proofing/.

[2] UK Government. Identity Proofing and Verification of an Individual. July 2014. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/370033/GPG_45_identity_proofing_v2_3_July_2014.pdf.

[3] William E. Burr, Donna F. Dodson, Elaine M. Newton, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, and Emad A. Nabbus. NIST SP 800-63-2 Electronic Authentication Guideline, August 2013.

http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf.

[4] Paul A. Grassi, James L. Fenton, Naomi B. Lefkovitz, Jamie M. Danker, Yee-Yin Choong, Kristen K. Greene, and Mary F. Theofanos. DRAFT NIST Special Publication 800-63A, Digital Authentication Guideline, Enrollment and Identity Proofing Requirements. Part A of public preview draft of SP 800-63-3. September 28 2016. https://pages.nist.gov/800-63-3/sp800-63a.html.

[5] IBM. Getting started with Identity Mixer (Experimental). https://console.ng.bluemix.net/docs/services/identitymixer/index.html.

[6] IBM Research, Zurich. Identity Mixer—A cryptographic algorithm to protect your privacy. http://www.research.ibm.com/labs/zurich/idemix/publications.html.

[7] Micosoft. U-Prove. https://www.microsoft.com/en-us/research/project/u-prove/.

[8] Francisco Corella and Karen Lewison. Privacy Postures of Authentication Technologies. http://pomcor.com/techreports/PrivacyPosturesMayFirstVersion.pdf.

[9] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2014.

[10] C. Rathgeb and A. Uhl. A Survey on Biometric Cryptosystems and Cancelable Biometrics. *EURASIP Journal on Information Security*, 3, 2011. http://jis.eurasipjournals.com/content/2011/1/3.

[11] Francisco Corella. Revocable Biometrics Discussion at the Internet Identity Workshop. May 2, 2016. https://pomcor.com/2016/05/02/revocable-biometrics-discussion-at-theinternet-identity-workshop/.

[12] F. Hao, R. Anderson, and J. Daugman. Combining Cryptography with Biometrics Effectively. *IEEE Trans. Comput.*, 55(9):1081–1088, 2006.

[13] Y. Desmedt, C. Goutier, and S. Bengio. Special Uses and Abuses of the Fiat-Shamir Passport Protocol. In *Advances in Cryptology - CRYPT0 '87, LNCS 293*, pages 21–39, 1987.

[14] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn.* Wiley, 1996.

[15] Elaine Barker, Lily Chen, Allen Roginsky, and Miles Smid. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST SP 800-56A Rev. 2. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf.

[16] W3C. Web Storage (Second Edition)—W3C Recommendation 19 April 2016. https://www.w3.org/TR/webstorage/.

[17] W3C. Service Workers—W3C Working Draft 25 June 2015. https://www.w3.org/TR/service-workers/.

[18] K. Kollreider, H. Fronthaler, M. I. Faraj, and J. Bigun. Real-Time Face Detection and Motion Analysis With Application in "Liveness" Assessment. *IEEE Transactions on Information Forensics and Security*, 2(3), September 2007.

[19] A. Turkmani. Visual Analysis of Viseme Dynamics. PhD Thesis, September 2007. http://epubs.surrey.ac.uk/804944/1/Turkmani2008.pdf.

[20] Sarah Hilder, Barry-John Theobald, and Richard W. Harvey. In pursuit of visemes. In *Auditory-Visual Speech Processing, AVSP 2010, Hakone, Kanagawa, Japan, September 30 - October 3, 2010*, pages 8–2, 2010. http://www.isca-speech.org/archive/avsp10/av10_S8-2.html.

[21] F. Corella and K. Lewison. It Is Time to Redesign Transport Layer Security. January 2014. https://pomcor.com/whitepapers/TimeToRedesignTLS.pdf.

[22] F. Corella and K. P. Lewison. Identity-Based Protocol Design Patterns for Machine-to-Machine Secure Channels, 2014. Presented at M2MSec'14. https://pomcor.com/techreports/M2MSec14.pdf.

[23] W3C. Web Cryptography API. https://www.w3.org/TR/WebCryptoAPI/.

[24] W3C. Indexed Database API. https://www.w3.org/TR/IndexedDB/.

[25] Dave Savini. 2 Investigators: China Floods U.S. With Near-Perfect Fake Driver's Licenses. September 22, 2014. http://chicago.cbslocal.com/2014/09/22/2-investigators-china-floods-u-s-with-near-perfect-fake-drivers-licenses/.