Using Trust Management to Support Transferable Hash-Based Micropayments

Simon N. Foley

Department of Computer Science,
University College, Cork, Ireland
s.foley@cs.ucc.ie

Abstract. A hash-chain based micropayment scheme is cast within a trust management framework. Cryptographic delegation credentials are used to manage the transfer of micropayment contracts between public keys. Micropayments can be efficiently generated and determining whether a contract and/or micropayment should be trusted (accepted) can be described in terms of a trust management compliance check. A consequence is that it becomes possible to consider authorisation based, in part, on monetary concerns. The KeyNote trust management system is used to illustrate the approach.

KEYWORDS: Delegation; Digital Cash; One-way Hash Functions; Public Key Certificates; Trust Management.

1 Introduction

Trust Management [3, 4, 9, 21] is an approach to constructing and interpreting the trust relationships between public keys that are used to mediate security critical actions. Cryptographic credentials are used to specify delegation of authorisation among public keys.

In this paper we consider how Trust Management can be used to manage trust relationships that are based on monetary payment. A benefit to characterising a payment scheme as a trust management problem is the potential to support, within the trust management framework, sophisticated trust requirements that can combine monetary and authorisation concerns. For example, authorisation to access a valuable resource might be based on some suitable combination of permission and monetary payment or deposit (possibly partially refundable if it can be determined that the resource is not misused). In this case, perhaps a user with 'less' permissions would be required to provide a larger deposit, while a user with 'more' permissions provides a smaller deposit. The trust management system is expected to help the application system manage what is meant by trusted access to this resource.

In [7, 14], the KeyNote trust management system is used to manage trust for a micro-billing based payment scheme. Their scheme is similar to IBM's mini-pay scheme [13]: KeyNote is used by the payer (merchant) to determine

whether or not an off-line payment from a particular payee (customer) should be trusted, or whether the payee should go online to validate the payment and payee. The scheme is intended for small value payments (under $1.00). Since the generation of each payment transaction requires a public key cryptographic operation (signature) then it may not be practical for very low value payments where the cost of processing is high relative to the value of the payment.

This paper extends earlier work [11] exploring how KeyNote can be used to to support low-value micropayments in the provision of authorised access to resources by the participants of a meta-computer The meta-computer [10, 18] is a network of heterogenous computers that work together to solve large problems. It encourages the sharing of resources between different organisations, departments and individuals. A consequence of this sharing is that the providers of the resources expect payment for their computation processing and services, on a per-use basis. For example, participants might be paid for contributing processing power to weather model computation.

In [11], we describe a preliminary KeyNote implementation of micropayments that is based on hash-chains [1, 8, 20]. These schemes are limited in that it is not possible for the participants to transfer (delegate) their micropayment contracts to third parties. Some form of efficient transfer is desirable. For example, an overloaded client workstation might like to transfer some of its workload and the corresponding micropayment contract that it holds to another client.

The contribution in this paper is the development of a hash-chain based micropayment scheme that is cast within a trust management framework and supporting the efficient transfer of micropayment contracts. Codifying a micropayment scheme in terms of KeyNote also demonstrates the usefulness and applicability of trust management systems in general.

The paper is organised as follows. Section 2 describes a simple model of cryptographic credential based delegation that forms the basis of trust management. Section 3 extends this model by considering how values (that represent permissions) along a hash-chain can be delegated without having to sign new credentials each time. This extension forms the basis of the transferable micropayment scheme that is proposed in Section 4. Section 5 considers how the scheme can be codified and interpreted within the KeyNote trust management system.

2 Delegating Authorisation

A simple model is used to represent delegation of authorisation between public keys. A signed cryptographic credential, represented as $\{\!|\, K_B, p \,|\!\}_{K_A}$, indicates that K_A delegates to K_B, the authorisation permission p. Permissions are structured in terms of lattice $(PERM, \leq, \sqcap)$, whereby $p \leq q$ means that permission q provides no less authorisation than p. A simple example is the power-set lattice of $\{\mathsf{read}, \mathsf{write}\}$, with ordering defined by subset, and greatest lower bound ($\sqcap$) defined by intersection.

Given a credential $\{\!|\, K_B, q \,|\!\}_{K_A}$ and $p \leq q$ then there is an implicit delegation of p to K_B by K_A, written as $(\!|\, K_B, p \,|\!)_{K_A}$. Two reduction rules follow.

$$\frac{\{\!|\, K_B, p \,|\!\}_{K_A}}{(\!|\, K_B, p \,|\!)_{K_A}} \text{ [R1]} \qquad \frac{(\!|\, K_B, p \,|\!)_{K_A}; p' \leq p}{(\!|\, K_B, p' \,|\!)_{K_A}} \text{ [R2]}$$

If delegation is regarded as transitive, and if K_A delegates p to K_B, and K_B delegates p to K_C, then it follows that K_A implicitly delegates p to K_C.

$$\frac{(\!|\, K_C, p \,|\!)_{K_B}; (\!|\, K_B, p' \,|\!)_{K_A}}{(\!|\, K_C, p \sqcap p' \,|\!)_{K_A}} \text{ [R3]}$$

This corresponds to SPKI certificate reduction [9] (greatest lower bound is equivalent to SPKI tuple intersection). It is not unlike a partial evaluation over a collection of KeyNote credentials, resulting in a single equivalent credential. Such reduction can be efficiently implemented using a depth first search of a delegation graph such as [2,6]. At this point, we do not consider permissions that cannot be further delegated and nor do we consider threshold schemes.

These rules are used by a trust management system to determine whether a request for an action (permission) is authorised. If the local policy specifies that K_A is authorised for p, and delegation $(\!|\, K_B, p \,|\!)_{K_A}$ can be deduced from some collection of credentials, then it follows that K_B is authorised for p.

3 Delegating Hash Chain Values

Consider the set of permission values *PERM* that are drawn from the range of a one-way cryptographic hash function $h()$. Section 4 provides one possible interpretation for such permissions that are based on hash-chain micropayments whereby each permission (a hash value micropayment) represents authorisation for payment.

Example 1 Principal K_A securely generates a secret random seed s and generates credential $\{\!|\, K_B, h(s) \,|\!\}_{K_A}$, delegating the 'permission' value $h(s)$ to K_B. With this credential, K_B is authorised for permission $h(s)$, but is not authorised for the permission s.

Suppose that at some later point in time, K_A wishes to delegate the permission s to K_B. Rather than having to sign a new credential $\{\!|\, K_B, s \,|\!\}_{K_A}$, K_A sends the value s over an unsecured channel to K_B. Principal K_B can then prove that it has been authorised by K_A for the permission s by presenting the original credential $\{\!|\, K_B, h(s) \,|\!\}_{K_A}$ and the value s: a third party simply checks that the hash of the s presented compares with the permission for which it holds a credential. The one-way nature of the hash function ensures that it is not feasible for K_B to forge the s permission before it is revealed. △

In general, a delegator K_A generates a secure random seed s and computes and issues credential $\{\!|\, K_B, h^n(s) \,|\!\}_{K_A}$. If the seed s is known only to K_A then

$[h^{n-1}(s), \ldots h^0(s)]$ forms a totally ordered chain of permissions that can be delegated to K_B by revealing the corresponding hash-value, one at a time, in order. Principal K_B is authorised for the permission x if $h^i(x)$ matches the value $h^n(s)$ in the original credential for some i known by the principal. We assume that the delegator K_A maintains a suitable interpretation for the semantics of the permissions it issues. The advantage of taking this approach is that K_A need sign only one initial delegation credential; subsequent delegations (along the permission chain, starting at the credential) do not require further costly cryptographic signatures. Since $h()$ is a one-way cryptographically secure hash function then it is not possible for the delegatee K_B to forge or to compute a permission that is yet to be delegated from the permission hash chain.

We provide an interpretation for this *hash-chain delegation* in terms of our simple model of delegation. This is done by defining an ordering relation over hash-value permissions $PERM$ as follows.

Definition 1 For $x, y \in PERM$ then $x \leq y$, with respect to a some principle, if and only if the principal knowing y can *feasibly determine* some x and i such that $h^i(x) = y$. △

Note that $x \leq y$ should not be taken to mean that a principal can reverse the hash function, rather, it means that the value has been revealed by a principal who knows the initial seed. In addition, note that the orderings in this relation increase (monotonically) to reflect what a principal can feasibly compute based on the hash permissions that it has received to date. For Example 1, delegatee K_B is in possession of $\{\!|\, K_B, y \,|\!\}_{K_A}$ where $y = h(s)$ and cannot feasibly find some x such that $h^i(x) = y$. However, once s has been revealed to K_B then it knows to calculate $h^1(s) = y$ and thus $s \leq y$.

Applying this definition of permission ordering to the certificate reduction rule R2 illustrates how revealing a hash permission generates an implicit delegation.

$$\frac{(\!|\, K_B, y \,|\!)_{K_A}; \text{can feasibly compute } h(x) = y}{(\!|\, K_B, x \,|\!)_{K_A}}$$

Example 2 Suppose that K_A generates a secret seed s and computes permissions $a = h^4(s)$, $b = h^3(s)$, $c = h^2(s)$ and $d = h^1(s)$. K_A first delegates permission a to K_B by writing $\{\!|\, K_B, a \,|\!\}_{K_A}$ and then reveals b and c (see Figure 1).

K_B can prove that it is authorised by K_A for permission c since it is authorised for a and it can feasibly compute $h(c) = b$ and and $h(b) = a$ and therefore, $c \leq b \leq a$, and by reduction $(\!|\, K_B, c \,|\!)_{K_A}$.

Suppose that K_B delegates authorisation for c to K_C by writing credential $\{\!|\, K_C, c \,|\!\}_{K_B}$. K_C can prove that it is authorised by K_A for c as follows. K_C holds credential chain $\{\!|\, K_B, a \,|\!\}_{K_A}$,$\{\!|\, K_C, c \,|\!\}_{K_B}$. Since it knows permission c it can feasibly calculate permissions $b = h(c)$ and $a = h(b)$ and therefore $c \leq b \leq a$. Thus, reduction rule R2 applied to $\{\!|\, K_B, a \,|\!\}_{K_A}$ gives implicit delegation $(\!|\, K_B, c \,|\!)_{K_A}$ and this, with credential $\{\!|\, K_C, c \,|\!\}_{K_B}$ gives $(\!|\, K_C, c \,|\!)_{K_B}$ by reduction rule R3, that is, K_C is authorised for c by K_A.

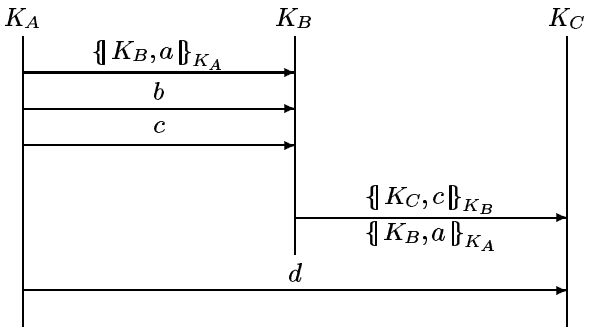

Fig. 1. Hash-Chain Delegation

Note that in this case it is not possible to use the reduction rules to prove that K_C is authorised for permissions a and b. To become authorised for these permissions it is necessary for K_B (or K_A) to explicitly delegate the permission by signing a suitable initial credential such as $\{\!|\, K_C, a \,|\!\}_{K_B}$.

K_A delegates permission d by making the value public, whereupon it becomes possible to deduce $(\!|\, K_C, d \,|\!)_{K_A}$ and $(\!|\, K_B, d \,|\!)_{K_A}$. △

This scheme is not limited to a single secret seed s: principals may generate and use as many seeds as they require. Assuming that seeds are generated in a cryptographically secure manner then the properties of the one-way hash function will ensure that collisions between permission chains are unlikely and that permission forgery is infeasible. When multiple seeds are used the permission ordering is composed of a series of independent chains (total orderings that can be feasibly generated) from each seed. In this case, the greatest lower bound operation $a \sqcap b$ is the lower of a and b when they are comparable (same chain). If they are incomparable (different chain) then the result of the operation is lattice 'bottom'.

If the manner of delegation is such that it can be structured according to a collection of hash-chains, as above, then we have an efficient way of performing delegation. Once the initial delegation credential $\{\!|\, K_B, y \,|\!\}_{K_A}$ is signed and issued then subsequent delegation of permissions along that chain do not require costly cryptographic signatures. The next section considers transferable hash-chain micropayments as a practical application of this general approach. Whether other applications of hash-chain delegation exist is a topic for future research.

In [16, 17] hash-chains are used to provide an efficient method for revocation of public key certificates. It can be interpreted within our framework as follows. With certificate $\{\!|\, K_B, [p, n, h^n(s)] \,|\!\}_{K_A}$, K_A delegates potential authorisation for permission p to K_B for n time-periods. Each hash value along the hash chain represents an authorisation for a particular time-period (starting with $h^n(s)$ for time period 0) In this case, K_A is considered authorised for time period i if a hash value v is presented such that $h^i(v) = h^n(s)$. At the start of a new time period, the certificate issuer makes available the corresponding hash value. Deciding not

to issue a hash-value provides an efficient form of revocation (at the granularity of the time-period). In [16,17] a hash-chain gives a authorisation time-line for a *single* permission. This differs slightly to our interpretation, where a hash-chain represents a particular total ordering over a set of permissions.

4 Transferable Hash Chain Micropayments

Hash-based micropayments schemes are intended to support very low-value payments and operate as follows.

A *payer* (the principal making the payment) securely generates a fresh random seed s, and computes $h^n(s)$, where $h()$ is a cryptographic one-way hash function. If s is known only to the payer, then $[h^{n-1}(s), n-1, val] \ldots [h^1(s), 1, val]$ provides an ordered chain of micropayments, each one worth val. Initially, the payer provides a *payee* with $[h^n(s), n, val]$, which acts as a contract for $(n-1)$ micropayments. It is required that the payer is unforgeably linked with the contract; for example, the payer signs the contract.

A payee (the principal receiving the payment) who has securely received i micropayments, $[h^{n-1}(s), n-1, val] \ldots [h^{n-i}(s), n-i, val]$, can use the hash function $h()$ to check their validity against the initial contract. Since $h()$ is a one-way hash function then it is not feasible for the payee to forge or compute the next $(i+1)^{th}$ payment (before it is issued). Micropayments may be cashed in/reimbursed by the payee at any time: the payment plus contract provides irrefutable evidence to a third party of the payer's obligation to honour the payment. To guard against double spending, the payer must keep track of irrefutable evidence of the reimbursements made to a payee against a contract.

This approach to micropayments has been proposed and used in payment schemes proposed by [1,8,20]. For example, in [1], the payer threads digital coins (issued by a bank) through the hash chain such that each micropayment reveals an authentic digital coin that can be reimbursed by the original bank.

In this paper we show how these schemes can, in general, be supported within a trust management system. Taking such an approach allows us to extend existing micropayment schemes by providing a framework that supports the transfer/delegation of micropayment contracts between principals.

Micropayments are interpreted in terms of trust management as follows. A payer sets up a contract by writing a suitable delegation certificate that contains the initial contract hash value. Hash-value micropayments correspond to (hash-value) permissions. A payee can cash in a micropayment if the payee has been (hash-chain) delegated its corresponding permission; this test can be done as a trust management compliance check.

Example 3 Consider Example 2. The payer K_A issues a new contract credential $\{\!|\, K_B, a \,|\!\}_{K_A}$. For the sake of simplicity, we assume that micropayment value (val) and the length of the hash chain (n) are universally agreed and fixed beforehand.

Two micropayments are made by K_A to K_B as b and c. Payee K_B can confirm that they are valid payments by carrying out a certificate reduction to

test $(\!| K_B, b |\!)_{K_A}$, and so forth. Payer K_A can validate a claim for reimbursement of a micropayment b from K_B by testing that K_B can prove $(\!| K_B, b |\!)_{K_A}$. We assume that K_A also checks for double spending. △

The proposed scheme allows a principal to transfer part of a micropayment contract to a third party. Suppose that a principal K_B holds a contract credential $\{\!| K_B, x |\!\}_{K_A}$ and has been paid up to micropayment y, where $x = h^i(y)$ and, therefore, implicitly holds $(\!| K_B, y |\!)_{K_A}$. Principal K_B can transfer the remainder of this contract to a third party K_C by signing $\{\!| K_C, y |\!\}_{K_A}$. In signing this credential, K_C is declaring that it gives up any claim that it originally held to seek reimbursement from K_A for micropayments subsequent to y, based on its original contract. The recipient K_C uses credential chain $[\{\!| K_B, x |\!\}_{K_A}, \{\!| K_C, y |\!\}_{K_B}]$ as a contract for any claims for reimbursement from K_A for micropayments subsequent to y.

Example 4 Continuing Example 3 (Figure 1), K_B transfers the remainder of its micropayment contract with K_A to K_C by writing $\{\!| K_C, c |\!\}_{K_B}$. K_C later claims for reimbursement of micropayment d from K_A by providing certificate chain $[\{\!| K_B, a |\!\}_{K_A}, \{\!| K_C, c |\!\}_{K_B}]$ as evidence of a valid contract along with d. As before, the validity of this claim is tested as a trust management compliance check for $(\!| K_C, d |\!)_{K_A}$. △

Note that the proposed scheme can be used to provide information to assist in the resolution of double payment on transferred contracts. Consider Example 4:

- If no claim for reimbursement for d has yet been made to K_A then that claim can be met.
- If K_A has already reimbursed K_C up to d and subsequently receives a claim for d from K_B then it will reject the claim. In this case it can provide the credential chain $[\{\!| K_B, a |\!\}_{K_A}, \{\!| K_C, c |\!\}_{K_B}]$ on the original claim to prove that K_B has given up the right to make such a claim.
- If K_A has already reimbursed K_B and it receives a claim from K_C then it will reject the claim. However it can provide $\{\!| K_C, d |\!\}_{K_B}$ to prove, in effect, that K_B had agreed not to make such a claim. In this case K_A or K_B may use this credential to seek restitution from K_B.

This scheme may be used to manage simple deposits, whereby a deposit is returned by transferring its original (unused) contract to the issuer. For example, K_A transfers an i micropayment deposit to K_B by signing contract $\{\!| K_B, h^n(s) |\!\}_{K_A}$ and revealing $h^{n-i}(s)$ (from which the chain of micropayments $h^{n-1}(s) \ldots h^{n-i}(s)$ may be calculated). The deposit recipient may return the deposit by signing $\{\!| K_A, h^n(s) |\!\}_{K_B}$ and returning it to K_A. As above, this credential provides evidence that K_B does not expect reimbursement for the i micropayments.

5 Towards a KeyNote Based Implementation

In this section we illustrate how the above scheme can be implemented within the KeyNote trust management system [3]. Note that we assume a version of

KeyNote with a minor modification, whereby we assume the ability to invoke an MD5 hash function within the KeyNote credential. Adding such functionality to the KeyNote Standard [3] would be trivial and would not impact on its architecture in any way.

We use the following general strategy. Suppose that K_A holds the top $h^i(s)$ of the hash chain

$$[h^i(s), h^{i-1}(s), \ldots h^{\texttt{ChainInd}}(s), \ldots h^0(s)]$$

K_A may have generated this hash chain (knows the seed s). Alternatively, $k^i(s)$ may be the most recent payment under some contract with another principle (public key) (who knows s) that K_A trusts. In this second case, 'trust' means that K_A trusts the contract issuer for the purposes of the particular contract.

If K_A issues $h^i(s)$ as a contract to K_B then it, in effect, authorises K_B for any hash value `HashVal` (that K_B can feasibly produce) in position `ChainInd` in the above chain such that

$$h^{i-\texttt{ChainInd}}(\texttt{HashVal}) = h^i(s) \tag{1}$$

holds. This authorisation is achieved in KeyNote by K_A writing a credential that conditionally delegates any value of `HashVal` such that the above holds, to K_B. Given particular values for the chain's length i and the chain's top $h^i(s)$, then K_A (delegating to K_B) signs a credential that includes a condition

$$h^{i-\texttt{ChainInd}}(\texttt{HashVal}) = h^i(s) \tag{2}$$

defined over attribute variables `ChainInd` and `HashVal`. In this case, K_B must know how to produce suitable values for `ChainInd` and `HashVal` such that the above holds (for a compliance check that K_B is authorised for the given `ChainInd` and `HashVal` to hold). Thus, the compliance check proves the validity of a payment [`ChainInd`,`HashVal`]. K_B cannot forge a payment since to do so would require reversing the one-way hash function.

Suppose that K_B has received j payments and wishes to delegate its remaining contract $h^{(i-j)}(s)$ to K_C. Given particular values for $h^{(i-j)}(s)$ and i and j then K_B (delegating to K_C) signs a credential that includes a condition

$$h^{(i-j)-\texttt{ChainInd}}(\texttt{HashVal}) = h^{(i-j)}(s) \tag{3}$$

defined over attribute variables `ChainInd` and `HashVal`. In this case, K_B must know how to produce suitable values for `ChainInd` and `HashVal` such that (3) above holds. If such values are known (by K_C), then K_C is authorised for [`ChainInd`,`HashVal`] by K_B. From (3), it follows that

$$\begin{aligned} h^j(h^{(i-j)-\texttt{ChainInd}}(\texttt{HashVal})) &= h^j(h^{(i-j)}(s)) \\ \Rightarrow h^{i-\texttt{ChainInd}}(\texttt{HashVal}) &= h^i(s) \end{aligned}$$

and, therefore, if equation (2) above holds in the credential issued by K_A to K_B, then K_C is also authorised for [`ChainInd`,`HashVal`] by K_A: a valid delegation

chain exists from K_A to K_C). This argument extends to any length delegation chain that is constructed in this manner. This general strategy is illustrated in the following examples.

We first consider how principals use KeyNote to determine whether they should trust (accept) contracts and subsequent micropayments from third parties.

Example 5 Consider Example 3. Principal `"Kb"` (Bob) trusts that `"Ka"` (Alice) will honour individual contracts that are worth up $5.00. This is expressed in terms of the following KeyNote credential.

```
Authorizer: "POLICY"
Licensees: "Ka"
Conditions: @PayVal * @ChainLen <= 500;
```

This *policy* credential defines the conditions under which micropayment contracts from the licensee key `"Ka"` are trusted by Bob. The conditions are defined using a C-like expression syntax in terms of the *action attributes* of the contract, that is, `PayVal` and `ChainLen`. Attribute `PayVal` gives the value of a single micropayment (in cents) and `ChainLen` is the length of the contract hash-chain. Note that the prefix '`@`' on a KeyNote attribute converts the string attribute variable to the integer value it represents. For the purposes of illustration we use names to represent public keys (and do not provide signatures).

Alice sends a request to Bob to accept a contract for up to four micropayments worth one cent each. The details of the contract are described in terms of attributes `PayVal` and `ChainLen`, as described above, and the attribute `HashVal` which gives the contact hash-value in position `ChainInd` = `ChainLen` (top of chain). In this case, Alice's contract is described by the following attribute bindings.

`PayVal` $\leftarrow 1$;
`ChainLen` $\leftarrow 4$;
`ChainInd` $\leftarrow 4$;
`HashVal` $\leftarrow$ `"PwhHLtyFH9yPUXEx4StCLA"`

Alice signs the following KeyNote credential for this contract.

```
Authorizer: "Ka"
Licensees:  "Kb"
Condition:  @PayVal==1 && @ChainLen==4 &&
  md5(@ChainLen-@ChainInd,HashVal)== "PwhHLtyFH9yPUXEx4StCLA";
Signature: ...
```

Operation `md5`(i, s) is the i^{th} iteration of the MD5 hash of s, that is, $h^i(s)$. In this credential `Ka` authorises `Kb` for any hash value payment `HashVal` in position `ChainInd` such that the condition holds. This condition is based on Equation (1) given at the start of Section 5. In the credential above, the contract hash-value is `md5`$(4, s)$ and is taken to correspond to the value a in Example 3.

Bob uses KeyNote to check whether he should trust this particular contract from Alice: he wishes to determine whether, under his own credential policy above, his key `Kb` has been suitably authorised by `Ka` for the attribute bindings defined above for the contract. Evaluating a KeyNote query for this request given the above credentials returns `true`, indicating a trusted delegation (of contract) from `Ka` to `Kb`.

Suppose that Alice sends the first payment `"uyMmZs0K7qgAKcJ+PAFYLw"`. Bob wishes check that this is an authorised payment, that is, it is a payment for which he can expect to be reimbursed. He verifies that `Kb` is authorised for this proposed payment (`HashVal`) by making a KeyNote query with the following attribute bindings.

`PayVal` $\leftarrow$ 1; (per contract)
`ChainLen` $\leftarrow$ 4; (per contract)
`ChainInd` $\leftarrow$ 3; (this first payment is in position 3)
`HashVal` $\leftarrow$ `"uyMmZs0K7qgAKcJ+PAFYLw"`; (value b from Example 3)

This query returns `true`: Bob now knows another hash value for which `Kb` is authorised by `Ka`.

Alice makes a second payment `"0CJkY1EEisC6OXOS36+jBA"` (corresponding to c in Example 3), which Bob checks by querying KeyNote with action attributes

`PayVal` $\leftarrow$ 1; `ChainLen` $\leftarrow$ 4. `ChainInd` $\leftarrow$ 2;
`HashVal` $\leftarrow$ `"0CJkY1EEisC6OXOS36+jBA"`.

and this also evaluates to true. $\triangle$

Bob now has two hash value micropayments that have been revealed by Alice and are, therefore, implicitly delegated by Alice via the initial contract credential. Bob presents these micropayments and the original contract credential and requests reimbursement from Alice. To determine whether the reimbursement request is valid, Alice must check that the micropayments are authorised based on the original contract credential. This check of validity by Alice is similar to the check carried out by Bob in the example above: given that Alice trusts her own key `Ka` then, a KeyNote query determines whether, given the contract credential, the key `Kb` authorised for the the given micropayments.

Note that KeyNote is used to verify that the micropayment is authorised in principle; it does not consider the potential for double spending, that is, multiple reimbursement requests for the same micropayment. Whether trust management would be useful in managing this is a topic for future research.

The next example illustrates how KeyNote is used to manage the general transfer of partially spent micropayment contracts.

Example 6 Continuing the previous example, suppose that Bob decides to transfer the remainder of his contract with Alice to Clare (`Kc`). As described in Example 4, this is done by Bob signing a credential that delegates authorization for the remaining hash chain of the contract to Clare.

The hash value $h^2(s)$ (hash value designated as c in the example) that Bob currently holds acts as a contract for the remainder of the hash chain that is controlled by Alice. Thus, applying Equation (1) of the general credential writing strategy, Bob signs the following contract credential.

```
Authorizer: "Kb"
Licensees: "Kc"
Condition:
   0<= @ChainInd && @ChainInd <=2 &&
   md5(2-@ChainInd,HashVal)== "0CJkY1EEisC6OXOS36+jBA";
Signature: ....
```

This authorises `Kc` for any `HashVal` (that she can feasibly produce) from the remaining hash chain ($2 \geq$ `ChainInd` ≥ 0).

As was similarly done by Bob, Clare can use KeyNote to check whether she should trust this contract. Assuming that Clare uses the same policy credential as Bob (trusting that `Ka` will honour contracts worth, at most, \$5.00) then she queries KeyNote to ensure that her key `Kc` is authorised for this contract described by attribute bindings:

`PayVal` $\leftarrow 1$; `ChainLen` $\leftarrow 4$. `ChainInd` $\leftarrow 2$;
`HashVal` $\leftarrow$ `"0CJkY1EEisC6OXOS36+jBA"`.

The credentials of Bob and Clare form a delegation chain from `Ka` to `Kc` authorising the contract hash value and thus the KeyNote query evaluates to `true`.

When Alice makes her third payment `"ICy5YqxZB1uWSwcVLSNLcA"` (corresponding to d in Example 4) uses KeyNote, as done by Bob, to check the validity of this `HashVal` in position `ChainInd=1`. Alice similarly processes a request for reimbursement from Clare by checking that a delegation chain exists from `Ka` to `Kc` for the particular payment. Recall from Section 4, that having made a contract transfer, then it is assumed that Bob is trusted not to claim for reimbursements of subsequent payments. If Alice decides that Bob is not trusted then it is straightforward in KeyNote for Alice to use a credential that authorizes Bob's contract such that the authorization cannot be further delegated/transferred. $\triangle$

One difficulty with the above KeyNote based scheme is that it generally requires repeated hash calculations when determining the validity of a micropayment. Consequently, the performance advantage of using a payment scheme based on one-way hash functions (instead of a public key based scheme such as [7]) diminishes as payments further down the hash chain are spent.

This performance penalty is avoided if the payment recipient compares, instead, the current hash payment (`HashVal`) against the previous payment (`PrevHash`) by checking that

$$\texttt{LastHash} == \texttt{md5}(1, \texttt{HashVal})$$

The following example illustrates how this check can be managed by the trust management system.

Example 7 Consider Example 5. Bob trusts that `Ka` will honour contracts worth up to 500 one cent micropayments. Bob offers service actions `X` and `Y`, charging 1 cent and 2 cents each time they are used, respectively. Bob's revised policy credential is specified as follows.

```
Authorizer: "POLICY"
Licensees: "Ka"
Conditions:
  Action=="NewContract" && @PayVal== 1 && @ChainLen<=500;
  Action=="X" -> {PrevHash==md5(1,HashVal) && @ChainInd>=0};
  Action=="Y" -> {PrevHash==md5(2,HashVal) && @ChainInd>=0};
```

Attribute `Action` defines the action requested: `"NewContract"` corresponds to a request to accept a new payment contract, and `"X"` and `"Y"` are the service actions offered.

When Bob receives a `"NewContract"` request from Alice he, as before, uses KeyNote to check that his own key `Kb` is suitably authorised under this policy. If the contract is accepted, then Bob stores the contract hash value as attribute `PrevHash`. When Alice requests service action `X`, Bob uses KeyNote to check that `Ka` is authorised for payment `HashVal`, given `Action` $\leftarrow$ `X`, etc. The compliance check succeeds when the payment upholds policy condition `PrevHash==md5(1,HashVal)` and Bob sets `PrevHash` to `HashVal`.

To use service `Y`, a hash-value that permits calculation of two payments must be presented by the payer. For example, if `PrevHash` is b (Example 2), then the `HashVal` presented is d, corresponding to payments d and $c = h(d)$, and policy condition `PrevHash==md5(2,HashVal)` is satisfied. $\triangle$

6 Discussion and Conclusion

This paper describes a hash-chain based micropayment scheme that is cast within a trust management framework and supports the efficient transfer of micropayment contracts. Cryptographic credentials are used to specify the transfer (delegation) of contracts between public keys. Determining whether a micropayment contract or payment should be trusted (accepted) corresponds to a trust management compliance check. A benefit to characterising the payment scheme in this way is that sophisticated trust requirements can be constructed both in terms of monetary and more conventional authorization concerns: access control can be based, in part, on the ability to pay. Codifying a micropayment scheme using KeyNote also demonstrates the usefulness and applicability of trust management systems in general.

The micro-billing scheme [7] uses KeyNote to help determine whether a micro-check (a KeyNote credential, signed by a customer) should be trusted and accepted as payment by a merchant. In [5] these microchecks provide a convenient way to manage the purchase of the hash-chain based coin stacks that are used in [19] to pay for wireless LAN/IEEE 802.11 access to public infrastructure. We believe that it would be straightforward to apply the technique described in

our paper to extend [5] to help (trust) manage the transfer of unspent portions of coin stacks between principals.

Hash-chains are used in [16, 17] to provide an authorization time-line for permission and provides an efficient method for revocation of public key certificates. It would be interesting to explore how our proposed strategy for coding hash-chains within KeyNote credentials could be applied to technique in [16, 17] to provide support for revocation of KeyNote credentials.

In [12, 15], hash functions are used to provide an efficient implementation of authentication. In this paper, hash functions are used to support an efficient form of delegation of authorization. Our proposed hash-chain delegation is restricted to orderings that can be characterised as disjoint collections of total orders (hash-chains) over permissions. Hash-chain micropayments are one example of this type of ordering. We are investigating how other techniques might be used to support a similar form of delegation of more general permissions.

Acknowledgements

Thanks to the anonymous reviewers for their useful comments on this paper and for drawing my attention to [16, 17]. Thanks also to Angelos Keromytis for his comments and for providing access to [5] prior to publication.

References

1. Ross Anderson, Harry Manifavas, and Chris Sutherland. Netcard - a practical electronic cash system. In *Cambridge Workshop on Security Protocols*, 1995.
2. Tuomas Aura. Comparison of graph-search algorithms for authorization verification in delegation networks. In *Proceedings of NORDSEC'97*, 1997.
3. M Blaze et al. The keynote trust-management system version 2. September 1999. Internet Request For Comments 2704.
4. M Blaze, J Feigenbaum, and J Lacy. Decentralized trust management. In *Proceedings of the Symposium on Security and Privacy*. IEEE Computer Society Press, 1996.
5. M. Blaze, J. Ioannidis, S. Ionnidis, A. Keromytis P. Nikander, and V. Prevelakis. Tapi: Transactions for accessing public infrastructure. submitted for publication, 2002.
6. Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210, 1999.
7. Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Offline micropayments without trusted hardware. In *Financial Cryptography*, Grand Cayman, February 2001.
8. Jean-Paul Boly et al. The ESPRIT project CAFE - high security digital payment systems. In *ESORICS*, pages 217–230, 1994.
9. C Ellison et al. SPKI certificate theory. September 1999. Internet Request for Comments: 2693.
10. Simon N. Foley, Thomas B. Quillinan, and John P. Morrison. Secure component distribution using WebCom. In *Proceeding of the 17th International Conference on Information Security (IFIP/SEC 2002)*, Cairo, Egypt, May 2002.

11. S.N. Foley and T.B Quillinan. Using trust management to support micropayments. In *In proceedings of the Annual Conference on Information Technology and Telecommunications*, Waterford, Ireland, October 2002.
12. Li Gong. Using One-way Functions for Authentication. *Computer Communication Review*, 19(5):8–11, 1989.
13. A. Herzherg and H. Yochai. Mini-pay: Charging per click on the web. In *Sixth International World Wide Web Conference*, 1997.
14. John Ioannidis et al. Fileteller: Paying and getting paid for file storage. In *Proceedings of Financial Cryptography*, March 2002.
15. Philippe A. Janson, Gene Tsudik, and Moti Yung. Scalability and flexibility in authentication services: The kryptoknight approach. In *INFOCOM (2)*, pages 725–736, 1997.
16. S. Micali. Efficient certificate revocation. In *Proceedings of the 1997 RSA Data Security Conference*, 1997.
17. S. Micali. Novomodo: Scalable certificate validation and simplified management. In *Proceedings of the First Annual PKI Research Workshop*, April 2002.
18. J.P. Morrison, D.A. Power, and J.J. Kennedy. A Condensed Graphs Engine to Drive Metacomputing. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA '99), Las Vagas, Nevada, June 28 - July1, 1999.
19. P. Nikander. Authorization and charging in public WLANs using FreeBSD and 802.1. In *Proceedings of the Annual USENIX Technical Conference, Freenix Track*, 2002.
20. Torben P. Pedersen. Electronic payments of small amounts. In *Security Protocols Workshop*, pages 59–68, 1996.
21. R Rivest and B Lampson. SDSI - a simple distributed security infrastructure. In *DIMACS Workshop on Trust Management in Networks*, 1996.