A Logic for Analysing Subterfuge in Delegation Chains

Hongbin Zhou and Simon N. Foley

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

Abstract. Trust Management is an approach to construct and interpret the trust relationships among public-keys that are used to mediate security-critical actions. Cryptographic credentials are used to specify delegation of authorisation among public keys. Existing trust management schemes are operational in nature, defining security in terms of specific controls such as delegation chains, threshold schemes, and so forth. However, they tend not to consider whether a particular authorisation policy is well designed in the sense that a principle cannot somehow bypass the intent of a complex series of authorisation delegations via some unexpected circuitous route.
In this paper we consider the problem of *authorisation subterfuge*, whereby, in a poorly designed system, delegation chains that are used by principals to prove authorisation may not actually reflect the original intention of all of the participants in the chain. A logic is proposed that provides a systematic way of determining whether a particular delegation scheme using particular authorisation is sufficiently robust to be able to withstand attempts at subterfuge. This logic provides a new characterisation of certificate reduction that, we argue, is more appropriate to open systems.

1 Introduction

Many commercial access control systems are closed and tend to rely on centralised authorisation policy/servers. An access control decision corresponds to determining whether some authenticated user has been authorised for the requested operation. This strategy of first determining who the user is and then whether that user is authorised has its critics, citing, for instance, single point of failure, scalability issues and excessive administrative overhead. A perhaps overlooked advantage of this approach is that administrators exercise tight control when granting access. The administrators are familiar with all of the resources that are available and they make sure that the user gets the appropriate permissions; no more and no less. The opportunity to subvert the intentions of a good administrator is usually small.

Cryptographic authorisation certificates bind authorisations to public keys and facilitate a decentralised approach to access control in open systems. Trust Management [15, 5, 9, 16, 1, 6] is an approach to constructing and interpreting the trust relationships among public-keys that are used to mediate access control. Authorisation certificates are used to specify delegation of authorisation among public keys. Determining authorisation in these systems typically involves determining whether the available certificates can prove that the key that signed a request is authorised for the requested action.

However, these approaches do not consider how the authorisation was obtained. They do not consider whether a principal can somehow bypass the intent of a complex series of authorisation delegations via some unexpected circuitous but authorised route. In an open system no individual has a complete picture of all the resources and services that are available. Unlike the administrator of the closed system, the principals of an open system are often ordinary users and are open to confusion and subterfuge when interacting with resources and services. These users may inadvertently delegate un-intended authorisation to recipients.

In this paper, we further explore the problem of *authorisation subterfuge* [14], whereby, in a poorly designed system, delegation chains that are used by principals to prove authorisation may not actually reflect the original intention of all of the participants in the chain.

For example, the intermediate principals of a delegation chain may inadvertently issue incorrect certificates, when the intended resource owner is unclear to intermediate participants in the chain. Existing Trust Management approaches such as [16] avoid this issue by assuming that all certificates are correctly in place, well understood by principals, and may not be improperly used.

However, we argue that subterfuge is a realistic problem that should addressed in a certificate scheme. For example, the payment systems [2, 3, 12] are vulnerable to authorisation subterfuge (leading to a breakdown in authorisation accountability) if care is not taken to properly identify the 'permissions' indicating the payment authorisations when multiple banks and/or provisioning agents are possible. In open systems, a permission for a resource should be uniquely related back to the resource owner, and this relationship should be understood by all related principals. If it is not well understood, then it may be subject to authorisation subterfuge. Therefore, authorisation in open systems should involve determining whether the available certificates can prove that the key that signed a request was intentionally authorised for the service.

In this paper we propose the Subterfuge Logic (SL) which can be used for analysing authorisation subterfuge. The logic is used to determine whether an authorisation through a delegation chain can be uniquely related to its intended resource and the resource owner.

The paper is organised as follows. In Section 2 we describe a series of subterfuge attacks that can be carried out on certificate chains. Section 3 explores similarities between these attacks on certificates and replay attacks on authentication protocols. Analysing a collection of certificates for potential subterfuge is not unlike checking whether it is possible for an 'intruder' to interfere with a certificate chain. Section 4 proposes the Subterfuge logic which can be used to determine whether performing a delegation operation might leave the delegator open to subterfuge. Examples from Section 2 are analysed in Section 5 and Section 6 illustrate how subterfuge can also arise in local naming. Finally, we conclude in Section 7.

2 Authorisation Subterfuge

2.1 SPKI/SDSI Authorisation

SPKI/SDSI [9] relies on the cryptographic argument that a public key provides a globally unique identifier that can be used to refer to its owner in some way. However, public keys are not particularly meaningful to users and, therefore, SPKI/SDSI provides local names which provide a consistent scheme for naming keys relative to another. For example, the local name that Alice uses for Bob is (Alice's Verisign's Bob), which refers to Bob's public key as certified by the Versign that Alice knows. By binding local names to public keys with name certificates, principals may delegate their authorisation to others beyond their locality through a chain of local relationships.

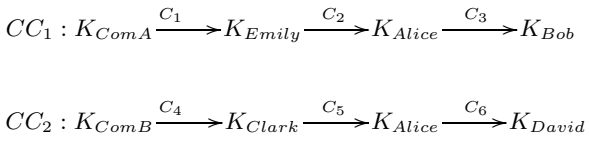
A SPKI/SDSI name certificate is denoted as (K, A, S), where: K specifies the certificate issuer's signature key, and identifier A is defined as the local name for the subject S. For example, $(K_B, \mathsf{Alice}, K_A)$ indicates that K_B refers to K_A using the local name Alice. A SPKI/SDSI authorisation certificate is denoted as (K, S, d, T), where: K specifies the certificate issuer's signature key; tag T is the authorisation delegated to subject S (by K) and d is the delegation bit (0/1). For example, K_B delegates authorisation T to Alice by signing $(K_B, \mathsf{Alice}, 0, T)$, where 0 indicates no further delegation. Note that for the sake of simplicity, in this paper, we do not include a validity period V in certificates.

Authorisation tags are specified as s-expressions and Example 2.6 in [10] specifies tag T1= tag (purchase(*range le <amount>),(*set <<items>>)) sych that it

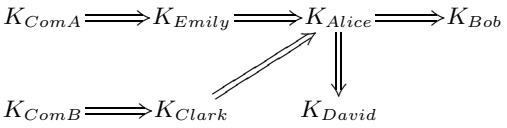
> *"[...] might indicate permission to issue a purchase order. The amount of the purchase order is limited by the second element of the (purchase) S-expression and, optionally, a list of purchasable items is given as the third element. The company*

whose purchase orders are permitted to be signed here will appear in the certificate permission chain leading to the final purchase order. Specifically, that company's key will be the issuer at the head of the (purchase). [...]" [10]

2.2 Authorisation Examples



(a) certificate chain CC_1 and CC_2



(b) delegation graph for T1

Fig. 1. Certificates in a Scenario

A company $ComA$ allows its manager $Emily$ to issue purchase orders, and $Emily$ may also delegate this right to others. After $Emily$ receives the certificate from $ComA$, $Emily$ delegates this right (issuing a purchase order) to an employee Bob via $Alice$. We have the following certificates: C_1=(K_{ComA}, K_{Emily}, 1, T1); C_2=(K_{Emily}, K_{Alice}, 1, T1), and C_3=(K_{Alice}, K_{Bob}, 0, T1) ($Alice$ delegates this right to employee Bob, But Bob may not delegate this right to others).

Suppose that there is another company $ComB$ which also uses the tag T1 to issue purchase orders. Suppose that Alice also works for $ComB$. Clark, a senior manager in $ComB$, holds the right to issue purchase orders, and delegate it to Alice. $ComB$ employee David accepts authority from Alice to issue purchase orders. We have certificates: C_4=(K_{ComB}, K_{Clark}, 1, T1); C_5=(K_{Clark}, K_{Alice}, 1, T1), and C_6=(K_{Alice}, K_{David}, 0, T1). Figure 1 gives the certificate chain CC_1 and CC_2 that Bob and David respectively use to prove authorisation to issue purchase orders.

2.3 Authorisation Subterfuge

The examples above are effective when separate chains CC_1 and CC_2 are used to prove authorisation. However, their combination, depicted in Figure 1(b), result in further delegation chains CC_3 and CC_4 and these lead to some surprising interpretations of how authorisation is acquired.

$$CC_3 : K_{ComA} \xrightarrow{C_1} K_{Emily} \xrightarrow{C_2} K_{Alice} \xrightarrow{C_6} K_{David}$$
$$CC_4 : K_{ComB} \xrightarrow{C_3} K_{Clark} \xrightarrow{C_4} K_{Alice} \xrightarrow{C_5} K_{Bob}$$

Subterfuge 1: passive attack. Alice's intention, when she signed C_6, was that David should use chain CC_2 as proof of authorisation when making purchases. However, unknown to Alice, dishonest David collects all other certificates and uses the chain CC_3 as his proof of authorisation.

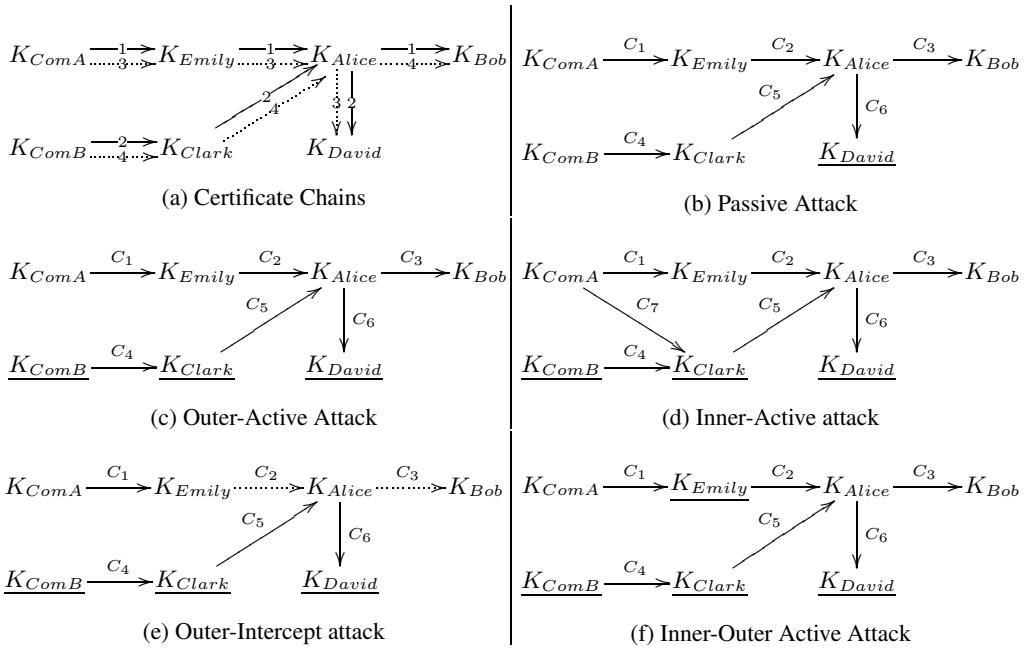


Fig. 2. Attack graphs

This confusion may introduce problems if the certificate chains that are used to prove authorisation are also used to provide evidence of who should be billed for the transaction. In delegating, Alice believes that chain CC_2 (from $ComB$) provides the appropriate accountability for ClarK's authorisation

Subterfuge 2: outer-active attack. The above passive attack can be transformed into a more active attack. David sets up a shelf company $ComB$ with fictitious employee Clark. Using attractive benefits, David masquerading as Clark, lures Alice to join $ComB$. Clark delegates authorisations ($\mathsf{T1}$) to Alice that correspond to authorisation already held by Alice. However, Alice does not realize this and, in the confusion, further delegates the authorisation to David; an authorisation from $ComA$ that normally he would not be expected to hold.

In both of these cases we think of Alice as more *confused* in her delegation actions rather than incompetent; the permission naming scheme influences her local beliefs and it was the inadequacy of this scheme that led to her confusion. Perhaps Alice has too many certificates to manage and in the confusion looses track of which permissions should be associated with which keys.

ComA may attack ComB in the same way to get the money back by CC_4. However, if ComB updates its certificate, then Alice does not hold the right for ComB. ComA cannot get its money back.

Subterfuge 3: inner-active attack. Clark is a manager in $ComA$ and $ComB$ and colludes with David ($ComB$ employee). Clark delegates authorisation $\mathsf{T1}$ legitimately obtained from $ComB$ to Alice. However, suppose that unknown to Alice, Clark is coincidentally authorised to do $\mathsf{T1}$ by $ComA$ (via C_7) and Clark intercepts the issuing of credential C_1 and conceals it. Alice delegates what she believes to be $\mathsf{T1}$ from $ComB$ to David via C_6. However, David can present chain $[C_7; C_5; C_6]$ as proof that his authorisation originated from $ComA$.

The above authorisation subterfuge may be avoided if Alice is very careful about how she delegates. However the following attacks are a bit more difficult for Alice to avoid.

Subterfuge 4: (outer-intercept attack) Clark intercepts certificate C_2 and conceals it. When delegating authorisation to David, Alice believes that the chain is $[C_4; C_5; C_6]$ from $ComB$, however David knowingly or unknowingly uses a different chain $[C_1; C_2; C_6]$.

Subterfuge 5: (inner-outer active attack). Alice has a legitimate expectation that so long as she delegates competently then she should not be liable for any confusion that is a result of poor system/permission design. Alice can use this view to act dishonestly. In signing a certificate she can always deny knowledge of the existence of other certificates and the inadequacy of permission naming in order to avoid accountability. While Alice secretly owns company $ComB$, she claims that he cannot be held accountable for the 'confusion' when Bob (an employee of $ComA$) uses the delegation chain $[C_4; C_5; C_3]$ to place an order for Alice.

2.4 Avoiding Subterfuge: Accounting for Authorisation

The underlying problem with the examples in the previous section is that the permission $\mathsf{T1}$ is not sufficiently precise to permit Alice to distinguish the authorisations that are issued by different principals. An ad-hoc strategy to avoid this problem would be to ensure that each permission is sufficiently detailed to avoid any ambiguity in the sense that it is clear from whom the authorisation originated. This provides a form of accountability for the authorisation. For example, including a company name as part of the permission may help avoid the vulnerabilities in the particular example above.

However, at what point can a principal be absolutely sure that an ad-hoc reference to a permission is sufficiently complete? Achieving this requires an ability to be able to fix a permission within a global context, that is, to have some form of global identifier and/or reference for the permission.

Public keys provide globally unique identifiers that are tied to the owner of the key. These can also be used to avoid permission ambiguity within delegation chains. For example, given SPKI authorisation certificate $(\mathsf{K}_{ComA}, \mathsf{K}_E, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$, there can be no possibility of subterfuge when Emily delegates to Alice with $(\mathsf{K}_E, \mathsf{K}_A, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$. In this case the authorisation $[\mathsf{T1}.\mathsf{K}_{ComA}]$ is globally unique and the certificate makes the intention of the delegation and where it came from (authorisation accountability) very clear.

SPKI [9] characterises the checking of authorisation as *"is principal X authorised to do Y?"*. However, the examples above illustrate that this is not sufficient; checking *"is principal X authorised to do Y by Y's owner Z?"* would be more appropriate.

Needless to say that this strategy does assume a high degree of competence on Alice's part to be able to properly distinguish between permissions $[\mathsf{T1}.\mathsf{K}_{ComA}]$ and $[\mathsf{T1}.\mathsf{K}_{ComB}]$, where, for example, each public key could be 342 characters long (using a common ASCII encoding for a 2048 bit RSA key). One might be tempted to use SDSI-like local names to make this task more manageable for Alice. However, in order to prevent subterfuge, permissions require a name that is unique across all name spaces where it will be used, not just the local name space of Alice. In Alice's local name space the permission $[\mathsf{T1}.$*(Emily's ComA)*$]$ may refer to a different $ComA$ to the $ComA$ that Alice knows.

Another possible source of suitable identifiers is a global X500-style naming service (if it could be built) that would tie global identities to real world entities, which would in turn be used within permissions. However, X500-style naming approaches suffer from a variety of practical problems [7] when used to keep track of the identities of principals. In the context of subterfuge, a principal might easily be confused between the (non-unique) common name and the global distinguished name contained within a permission that used such identifiers.

Certificate chains have been used in the literature to support degrees of accountability of authorisation, for example, [3, 12, 2]. The micro-billing scheme [3] 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. The originator of the chain is the provisioning agent, who is effectively responsible for ensuring that the transaction is

paid for. In [12], delegation credentials are used to manage the transfer of micropayment contracts between public keys; delegation chains provide evidence of contract transfer and ensure accountability for double-spending. These systems are vulnerable to authorisation subterfuge (leading to a breakdown in authorisation accountability) if care is not taken to properly identify the 'permissions' indicating the payment authorisations when multiple banks and/or provisioning agents are possible.

3 Subterfuge in Satan's Computer

Authorisation subterfuge is possible when one cannot precisely account for how an authorisation is held. In signing a certificate, we assume that the signer is in some way willing to account for the authorisation that they are delegating. The authorisation provided by a certificate chain that is not vulnerable to subterfuge can be accounted for by each signer in the chain. A principal who is concerned about subterfuge will want to check that the permission that is about to be delegated can also be accounted for by others earlier in the chain: the accountability 'buck' should preferably stop at the head of the chain!

We are interested in determining whether, given a collection of known certificates, it is safe for a principal to delegate some held authorisation to another principal. By safe we mean that subterfuge is not possible. In simple terms, this requires determining if it is possible for a malicious outsider to interfere with a certificate chain with a view to influencing the authorisation accountability. In order to help understand this we draw comparisons between subterfuge attacks and attacks on authentication protocols. Our hypothesis is that techniques for analysing one can be used to analyse the other (as we shall see in the next section when we use a BAN-like logic to analyse subterfuge in delegation chains).

A certificate is a signed message that is exchanged between principals; an authentication protocol step can be an encrypted message that is exchanged between principals. A certificate chain is an ordering of certificates exchanged between principals. An authentication protocol is an ordering of encrypted messages exchanged between principals. For example, the chain CC_1 could be represented by the following protocol.

$$\begin{array}{lll} msg1 & ComA \rightarrow E & : \{K_{ComA}, K_E, 1, T1\}_{K_{ComA}} \\ msg2 & E \rightarrow A & : \{K_E, K_A, 1, T1\}_{K_E} \\ msg3 & A \rightarrow B & : \{K_A, K_B, 0, T1\}_{K_A} \end{array}$$

There are differences between authentication protocols and certificate chains. A round of a typical authentication protocol has a fixed and small number of pre-defined messages, while the number of participants and messages in a certificate chain are unlimited and, sometimes, it may not be predetermined.

An attack from Section 2 is represented as follows.

$$\begin{array}{lll} msg2'. & I(CA) \rightarrow A & : \{K_I, K_A, 1, T1\}_{K_I} \\ msg3'. & A \rightarrow D & : \{K_A, K_D, 0, T1\}_{K_A} \end{array}$$

Subterfuge attacks involve a malicious user (the intruder I) removing/hiding and replaying certificates between different certificate chains. These actions are comparable to a combination of the replay attacks [4]:

Freshness attack "When a message (or message component) from a previous run of a protocol is recorded by an intruder and replayed as a message component in the current run of the protocol."

Parallel session attack "When two or more protocol runs are executed concurrently and messages from one are used to form messages in another."

The analysis of an authentication protocol typically centres around an analysis of nonce properties: *if one may correctly respond to the nonce challenge in a round of an authentication protocol, it is the regular responder.*

Freshness A nonce is a number used once in a message. Message freshness fixes a message as unique and ties it to a particular protocol run.

Relevancy to originator A nonce is related to its originator. The nonce verifier is also the nonce provider (originator). The nonce originator generates the nonce and this means that it can recognise and understand its relationship with the nonce.

Relevance of message In a two-party mutual authentication protocol, each principal generates its own nonce. A principal uses its own nonce and the other principal's nonce to relate its own message to the other's message.

There are some similarities between these nonce properties and the permission properties that rely on unique permissions.

Uniqueness is required in a permission string to account for its originator within a particular certificate chain.

Relevancy to originator A permission should be related to its originator and it should be possible for others along the chain to recognise this relationship.

Relevance of certificates Certificates can be used to delegate combinations of permissions that originated from different sources. These new certificates should be account for the authorisation of the originators.

Lowe [17] defines the correctness of authentication as:

> *"A protocol guarantees agreement to a participant B (say, as the responder) for certain data items x if: each time a principal B completes a run of the protocol as responder using x, which to B appears to be a run with A, then there is a unique run of the protocol with the principal A as initiator using x, which to A appears to be a run with B."*

We characterise accountability of authorisation within a certificate chain as follows.

> A certificate chain guarantees the principal A's accountability of authorisation to a participant B (say, as the delegatee) for certain permission R if: each time a principal B is delegated a right R, which to B appears to be a certificate chain with A, then there is a unique certificate chain with the principal A as initial delegator authorising R.

We use a BAN-style logic to reason about this notion of accountability of authorization.

4 A Logic for Analysing Certificate Chains

In the last thirty years, a variety of techniques for analysing authentication protocols have been proposed. The previous section demonstrated similarities between (freshness) vulnerabilities in authentication protocols and (subterfuge) vulnerabilities in delegation chains. In this section we develop the Subterfuge Logic (SL) which draws on some of the techniques from BAN-like logics to analyse subterfuge in certificate chains.

4.1 The language

The logic uses the following basic formulae. P, Q,R and S range over principals; X represents a message, which can be data or formulae or both; ϕ will be used to denote a formula. The basic formulae are the following:

- $\sharp(X)$: Formula X is a globally unique identifier. For example, this is typically taken as true for X.500 distinguished names and for public keys.
- $X \mid P$: represents the message X, as guaranteed/accounted for by principal P; this means that P is willing to be held accountable for the consequences of action X. For example, it is in Alice's interest to delegate $\mathsf{T1} \mid K_{ComA}$ to Bob, as opposed to just $\mathsf{T1}$.

– $X \rightsquigarrow P$: Principal P is an originator of formula X. In the examples above, we write $\mathsf{T1}|K_{ComA}$ to mean that permission $\mathsf{T1}$ was first uttered by K_{ComA} in some chain. Note that we assume that the same global unique formula (permission) cannot originate from two different principals, that is, if $X \rightsquigarrow P$, $X \rightsquigarrow Q$ and $\sharp(X)$ then $P = Q$.
– $P \ni X$: P is authorised for the action X.
– $P \succ X$: P is authorised to delegate X to others.
– $P \mid\!\sim X$: P directly says X. This represents a credential that is directly exchanged between principals.
– $P \mid\sim X$: P says X. P directly says X or others say X (who have been delegated to speak on X by P).

Further formulae can be derived by using propositional logic. If ϕ_1 and ϕ_2 are formulae, then $\phi_1 \wedge \phi_2$ (ϕ_1 and ϕ_2), $\phi_1 \vee \phi_2$ (ϕ_1 or ϕ_2), and $\phi_1 \rightarrow \phi_2$ are formulae.

SPKI/SDSI credentials can be encoded within the logic as follows. An authorisation credential $(\mathsf{K}, \mathsf{S}, 0, \mathsf{T})$ is represented as $\mathsf{K} \mid\!\sim (\mathsf{S} \ni \mathsf{T})$, and credential $(\mathsf{K}, \mathsf{S}, 1, \mathsf{T})$ represented as $\mathsf{K} \mid\!\sim (\mathsf{S} \ni \mathsf{T} \wedge \mathsf{S} \succ \mathsf{T})$. The purpose of the logic is to permit a principal decide whether it would be safe for it to delegate an authorisation based on the collection of credentials that it currently holds. For the examples above, Alice would like to be able to test whether it is safe for her to write a credential corresponding to $K_{Alice} \mid\!\sim (K_{David} \ni \mathsf{T1})$. That is, she wishes that someone further back on the chain will accept accountability for the action, that is, $K_{Alice} \succ \mathsf{T1}|K_{ComA}$ can be deduced (which is not possible for the examples in Section 2). Note that in signing the credential, Alice is also accepting accountability for the authorization.

4.2 Inference rules

Gaining Rules

G1 If P holds authorisation for X, for which Q can be held accountable, and Q may delegate X then P is also authorised for X.

$$\frac{P \ni X \,|\, Q, Q \succ X}{P \ni X}$$

G2 We have a similar rule for authorisation to delegate.

$$\frac{P \succ X \,|\, Q, Q \succ X}{P \succ X}$$

Direct delegation

D1 Direct delegation of authority assumes that the delegator accepts responsibility for the action.

$$\frac{P \mid\!\sim (Q \ni X)}{P \mid\sim (Q \ni X \,|\, P), Q \ni X \,|\, P}$$

D2 We have a similar rule for authorisation to delegate.

$$\frac{P \mid\!\sim (Q \succ X)}{P \mid\sim (Q \succ X \,|\, P), Q \succ X \,|\, P}$$

D3 The usual conjunction rules apply.

$$\frac{P \mid\!\sim (\phi_1 \wedge \phi_2)}{P \mid\!\sim \phi_1, P \mid\!\sim \phi_2}$$

Indirect delegation

I1 If principal P says that Q is authorised to perform an action X (with R accountable), and P is authorised to delegate X (with R accountable), then Q is authorised to perform X (with R accountable).

$$\frac{P \mid\sim (Q \ni X \,|\, R), P \succ X \,|\, R}{Q \ni X \,|\, R}$$

I2 We have a similar rule for authorisation to delegate.

$$\frac{P \mid\sim (Q \succ X \,|\, R), P \succ X \,|\, R}{Q \succ X \,|\, R}$$

I3 If principal P says that Q is authorised to perform action X by P, then P says that Q is authorised to perform X.

$$\frac{P \mid\sim (Q \ni X \,|\, P)}{P \mid\sim (Q \ni X)}$$

I4 Accountability can be stripped from an authorisation. Note, however, that stripping accountability does not refute the existence of the accountability.

$$\frac{P \mid\sim (Q \succ X \,|\, P)}{P \mid\sim (Q \succ X)}$$

I5 Accountability is transitive along certificate chains.

$$\frac{P \mid\sim (Q \succ X \,|\, R), R \mid\sim (P \succ X \,|\, S)}{R \mid\sim (Q \succ X \,|\, S)}$$

I6 We have a similar rule for authorisation.

$$\frac{P \mid\sim (Q \ni X \,|\, R), R \mid\sim (P \succ X \,|\, S)}{R \mid\sim (Q \ni X \,|\, S)}$$

Unique Origin Rules

U1 If Q is authorised for unique X that originated from P then P can be held accountable for X.

$$\frac{\sharp(X), X \rightsquigarrow P, Q \ni X}{Q \ni X \,|\, P}$$

U2 We have a similar rule for authorisation to delegate.

$$\frac{\sharp(X), X \rightsquigarrow P, Q \succ X}{Q \succ X \,|\, P}$$

5 Analysing Authorisation Subterfuge

The example from Section 2 is analysed using the Subterfuge Logic as follows. Certificates C_1 and C_2 are encoded by the following formulae. Note that principal names are abbreviated to their first initial if no ambiguity can arise.

$$K_{ComA} \mid\sim ((K_E \ni \mathsf{T1}) \wedge (K_E \succ \mathsf{T1}))$$
$$K_E \mid\sim ((K_A \ni \mathsf{T1}) \wedge (K_A \succ \mathsf{T1}))$$

Assumptions regarding uniqueness include the following.

$$\sharp(K_{ComA}), \sharp(K_{ComB}), \sharp(K_A), \sharp(K_B), \sharp(K_C), \sharp(K_E)$$

Principal $ComA$ is assumed authorised to delegate and accept accountability for the authorisations $\mathsf{T1}$ that it originates.

$$K_{ComA} \succ (\mathsf{T1} \,|\, K_{ComA})$$

Before delegating authority for $\mathsf{T1}$ to Bob, Alice wishes to test whether it is safe to do so. Alice tests whether $ComA$ accepts accountability for this action, that is she attempts to deduce $K_A \succ \mathsf{T1} \mid K_{ComA}$ using the above assumptions within the logic. This is not possible since no assumption is made regarding uniqueness of $\mathsf{T1}$, and, therefore, we cannot deduce $K_E \mid\!\sim (K_A \succ T_1 \,|\, K_{ComA})$; thus Alice refrains from the delegation.

In Trust Management public keys provide globally unique identifiers that are tied to the owner of the key. These can also be used to avoid authorisation ambiguity within delegation chains. For example, given SPKI certificate $(\mathsf{K}_{ComA}, \mathsf{K}_E, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$, there can be no possibility of subterfuge when Emily delegates to Alice by signing the certificate $(\mathsf{K}_E, \mathsf{K}_A, 1, [\mathsf{T1}.\mathsf{K}_{ComA}])$. In this case the authorisation $[\mathsf{T1}.\mathsf{K}_{ComA}]$ is globally unique, that is $\sharp(\mathsf{T1}|\mathsf{K}_{ComA})$ and the certificate makes the intention of the delegation and accountability very clear.

The revised certificates are represented in the logic as follows.

$$\begin{aligned} K_{ComA} &\mid\!\sim ((K_E \ni T_1 \,|\, K_{ComA}) \wedge (K_E \succ T_1 \,|\, K_{ComA})) \\ K_E &\mid\!\sim ((K_A \ni T_1 \,|\, K_{ComA}) \wedge (K_A \succ T_1 \,|\, K_{ComA})) \end{aligned}$$

Given these certificates then Alice can deduce

$$K_A \succ \mathsf{T1} \,|\, K_{ComA}$$

and can safely delegate to Bob as

$$K_A \mid\!\sim (K_B \ni T_1 \,|\, K_{ComA})$$

and we can deduce that $K_B \ni T_1 \,|\, K_{ComA}$. Considering other certificates, including

$$\begin{aligned} K_{ComB} &\mid\!\sim ((K_C \ni T_1 \,|\, K_{ComB}) \wedge (K_C \succ T_1 \,|\, K_{ComB})) \\ K_C &\mid\!\sim ((K_A \ni T_1 \,|\, K_{ComB}) \wedge (K_A \succ T_1 \,|\, K_{ComB})) \\ K_A &\mid\!\sim (K_D \ni T_1 \,|\, K_{ComB}) \end{aligned}$$

we can deduce $K_D \ni T_1 \,|\, K_{ComB}$, the expected authorisation.

Suppose that $ComB$ issues confusing certificates to Clark, who in turn delegates the incorrect authorisation to Alice.

$$\begin{aligned} K_{ComB} &\mid\!\sim ((K_C \ni T_1 \,|\, K_{ComA}) \wedge (K_C \succ T_1 \,|\, K_{ComA})) \\ K_C &\mid\!\sim ((K_A \ni T_1 \,|\, K_{ComA}) \wedge (K_A \succ T_1 \,|\, K_{ComA})) \end{aligned}$$

In this case we can deduce $K_{ComB} \mid\!\sim (K_A \succ T_1 \mid K_{ComA})$ and thus and $K_A \succ T_1 \mid K_{ComB}$. However, before A delegates this right for K_{ComA}, she needs (but cannot hold) the following formulae $K_{ComB} \succ T_1 \,|\, K_{ComA}$, or $K_C \succ T_1 \,|\, K_{ComA}s$. Thus, she should not delegate and therefore resists the subterfuge attack.

The conventional SPKI/SDSI authorisation certificate reduction rule can be described as

$$P \mid\!\sim (Q \succ X) \wedge Q \mid\!\sim (R \succ X) \rightarrow P \mid\!\sim (R \succ X)$$

in the SL logic (with a similar relationship for delegation of authorisation). Such relationship does not facilitate the tracking of accountability during certificate reduction.

6 Subterfuge in Local Names

Subterfuge is also possible when using local name certificates. Ellison and Dohrmann [8] describe a model based on SPKI/SDSI name certificates for access control in mobile computing platforms. A group leader controls all rights of a group. A group leader may delegate the right "admitting members" to other principals. For example, K_G is a group leader; K_G admits K_A as its group member by certificate C_1. K_G defines a large random number n, which will be used as K_A's local name for K_G's member. Then, K_G issues certificate C_2 to K_A which means that if K_A accepts a principal as (K_A's n), then the principal also becomes K_G's group G's member. K_A admits K_B as K_A's n by C_3. Together with C_2, K_B also becomes a member of K_G's G as presented in C_4. The certificates are as follows.

$$C_1 = (\mathsf{K_G}, \mathsf{G}, \mathsf{K_A}); \quad C_2 = (\mathsf{K_G}, \mathsf{G}, (\mathsf{K'_A s\ n})); \quad C_3 = (\mathsf{K_A}, \mathsf{n}, \mathsf{K_B})$$

From these we can deduce $(\mathsf{K_G}, \mathsf{G}, \mathsf{K_B})$, that is, $\mathsf{K_B}$ is now a member of group G.

The scheme works in a decentralised manner and thus no single member will hold the entire membership list. This means that there is no easy way to prove non-membership. The strategy described in the paper is sufficiently robust as it relies on face-to-face verification of certificate C_2 when a member joins.

However, the nonce is large and there may be potential for confusion during the face-to-face verification and this can lead to subterfuge. Consider the following certificates.

$$C'_1 = (\mathsf{K_I}, \mathsf{G_I}, \mathsf{K_A}); \quad C'_2 = (\mathsf{K_I}, \mathsf{G_I}, (\mathsf{K'_A s\ n})); \quad C'_3 = (\mathsf{K_A}, \mathsf{n}, \mathsf{K_I})$$

Suppose that the intruder K_I wants to join K_G's group G. K_I intercepts C_2 and issues C'_2 by using the number in C_2. In the confusion, K_A issues C'_3 which corresponds to admitting K_C (which the intruder controls) as a member of K_I's G_I for K_A. In this case, K_C may use C_2 and C'_3 to prove its membership in K_G's group G.

7 Conclusions

In this paper we described how poorly characterised permissions within cryptographic credentials can lead to authorisation subterfuge during delegation operations. This subterfuge results in a vulnerability concerning the accountability of the authorisation provided by a delegation chain: does the delegation operations in the chain reflect the true intent of the participants?

The challenge here is to ensure that permissions can be referred to in a manner that properly reflects their context. Since permissions are intended to be shared across local name spaces then their references must be global. In the paper we discuss some ad-hoc strategies to ensure globalisation of permissions. In particular, we consider the use of global name services and public keys as the sources of global identifiers.

The Subterfuge Logic proposed in this paper provides a systematic way of determining whether a particular delegation scheme using particular ad-hoc permissions is sufficiently robust to be able to withstand attempts at subterfuge. This logic provides a new characterisation of certificate reduction that, we argue, is more appropriate to open systems. We believe that it will be straightforward to extend the Subterfuge Logic to consider subterfuge in SDSI-like local names (as considered in Section 6).

Trust Management, like many other protection techniques, provide operations that are used to control access. As with any protection mechanism the challenge is to make sure that the mechanisms are configured in such a way that they ensure some useful and consistent notion of security. Subterfuge logic helps to provide assurance that a principal cannot bypass security via some unexpected but authorised route. This general goal of analysing unexpected but authorised access is not limited to just certificate schemes. Formal techniques that analyse whether a particular configuration of access controls is effective is considered

in [11, 13]; strategies such as well formed transactions, separation of duties and protection domains help to ensure that a system is sufficiently robust to a malicous principle. We are currently exploring how the subterfuge logic can be extended to include such robustness building strategies.

8 Acknowledgements

This work is supported by the UCC Centre for Unified Computing under the Science Foundation Ireland WebComG project and by Enterprise Ireland Basic Research Grant Scheme (SC/2003/007).

References

1. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The keynote trust-management system, version 2, September 1999. IETF RFC 2704.
2. M. Blaze, J. Ioannidis, S. Ioannidis, A. Keromytis, P. Nikander, and V. Prevelakis. Tapi: Transactions for accessing public infrastructure. In *Proceedings of the 8th IFIP Personal Wireless Communications (PWC) Conference*, 2003.
3. M. Blaze, J. Ioannidis, and A. D. Keromytis. Offline micropayments without trusted hardware. In *Financial Cryptography*, Grand Cayman, February 2001.
4. J. A. Clark and J. L. Jacob. A survey of authentication protocol literature, version 1.0. In *http://www.cs.york.ac.uk/jac/*, 1997.
5. D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in spki/sdsi. *Journal of Computer Security*, 9(4):285–322, 2001.
6. J. DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Research in Security and Privacy*, pages 105–113. IEEE Computer Society Press, 2002.
7. C. Ellison. The nature of a usable PKI. *Computer Networks*, 31:823–830, 1999.
8. C. Ellison and S. Dohrmann. Public-key support for group collaboration. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):547–565, 2003.
9. C. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. Thomas, and T. Ylonen. Spki certificate theory, September 1999. IETF RFC 2693.
10. C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. Spki examples, September 1998.
11. S. Foley. A non-functional approach to system integrity. *IEEE Journal on Selected Areas in Communications*, 21(1), Jan 2003.
12. S. Foley. Using trust management to support transferable hash-based micropayments. In *Proceedings of the 7th International Financial Cryptography Conference*, Gosier, Guadeloupe, FWI, January 2003.
13. S. Foley. Believing in the integrity of a system. In *IJCAR Workshop on Automated Reasoning for Security Protocol Analysis*. Springer Verlag Electronic Notes in Computer Science, 2004.
14. S. N. Foley and H. Zhou. Authorisation subterfuge by delegation in decentralised networks. In *International Security Protocols Workshop*, Cambridge, UK, April 2005.
15. R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, April 2002.
16. N. Li et al. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of 2003 IEEE Symposium on Security and Privacy*. IEEE, 2003.
17. G. Lowe. A hierarchy of authentication specifications. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.