DIVISION OF COMPUTER SCIENCE

Nested Signature Blocks

Marie Rose Low
Bruce Christianson

Technical Report No.223

March 1995

Nested Signature Blocks

Marie Rose Low and Bruce Christianson
M.R.Low@herts.ac.uk B. Christianson@herts.ac.uk

Abstract

For any signature block and any other data, there exists a key which produces the same signature block. In this report we identify the threat that this poses for the SAProxy scheme which uses nested signature blocks as pointers to other tokens. A modification to public key certificates is then proposed to eliminate this threat.

1. The Threat

A signature block for a set of data is generated (in simple terms) by encrypting the data with a particular key such that

$\{D\}K=C$

where C is the cyphertext = signature.

For any cyphertext, C, produced as above, and for any other data, D', there exists an easily computed key, K', such that

$\{D'\}K' = C$

It therefore appears that referring to signed data by its signature block leaves the way open for a perpetrator to use a valid signature block to refer to false data signed under a different key but yielding the same signature block. In schemes, such as the SAProxy scheme [LoCh94a] [LoCh94b], signature blocks are used to refer both to public key certificates (PKC) and other SAProxies and it seems that the above observation poses a threat which may undermine the validity of the scheme.

2. Public Key Certificates

The threat is most pertinent to PKCs. A PKC for B, C_B, is usually of the form:

C_B :

B	K^+_B	Cid	life-span	$sig(C_B)K^-_C$

where B is the identity of the owner of the public key, K^+_B, (and hence the PKC), Cid is the identity of the certification authority (CA) which generated the PKC and K^-_C is the CA's private key.

A perpetrator, X, then generates a false PKC, C_X, with a public key, K^+_X, whose matching private key, K^-_X, is known to X. X chooses a new, false key pair for C, $K^+_{C'}$ and $K^-_{C'}$, and generates C_X.

C_X :

X	K^+_X	Cid	life-span	$sig(C_X)K^-_{C'}$

$K^-_{C'}$ is chosen so that $\mathbf{sig(C_X)K^-_{C'} = sig(C_B)K^-_C}$.

X is then able to use any of B's SAProxies that use $sig(C_B)K^-_C$ as a reference to B's

PKC. X can make a request or delegate authority to another party by sending C_X with the request signed using K^-_X.

The problem now left to X is to persuade the verifier of the request that C's public key is now $K^+_{C'}$ and not K^+_C. If the verifier's procedures to obtain C's public key are not very stringent or if X violates the party(ies) trusted by the verifier to supply him with this key, then X will get away with the transaction and successfully use B's SAProxies.

The greatest risk may appear to arise when a trusted principal is endorsing another principal's token e.g. a visa or Freshness Certificate [Low94], and so accepting liability for that token. Consider the situation where the verifier knows and trusts A who endorses C_B with a visa. X may then send this visa with C_X to the verifier. The verifier believes that C_X is valid because it is 'endorsed' by A. However, the verifier must still check the PKC's, C_X, signature and in doing so must use $K^+_{C'}$ if C_X is to validate correctly. Principal A has not endorsed $K^+_{C'}$, so the verifier has to be persuaded of this key's validity by some other means. The responsibility for getting the correct CA public key still rests with the verifier.

N.B.

1. X does not have to violate the CA, C, to do this. X has to use a different CA key.
2. X does not have to pretend to be B in the false PKC, though of course this is also possible.
3. Generating such a false PKC is not possible with the private key of a violated CA i.e. a different key is necessary to generate the same signature for different data. Therefore violating a CA does not give rise to this situation.

2.1 Solutions

The problem can be tackled in two ways:

a. Instead of nesting the signature of the delegatee's PKC in a token, the delegatee's public key is included. Then only the principal with knowledge of the matching private key can make use of the token. The remaining problem is how to identify the real owner of the public key.

b. The second solution is to make it computationally infeasible to produce a pre-determined signature block. This is achieved by including the public key of the CA in the PKCs.

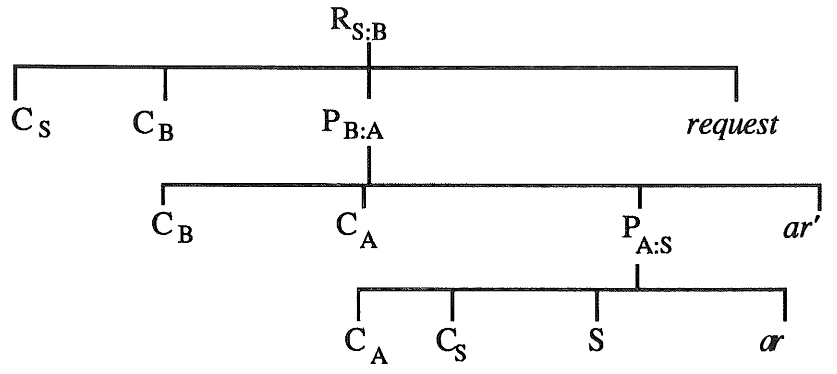
C_B :						
	B	K^+_B	Cid	K^+_C	life-span	sig(C_B)K^-_C

For every attempt at finding a bogus CA private key which generates the same PKC signature as another, the value of the corresponding public key included in the PKC, i.e. the data to be signed, also changes. Thus by this addition of the CA's public key, it becomes infeasible to generate a false PKC with a particular signature and thus the threat to PKCs may be removed.

3. Tokens of Delegation

Are the tokens of delegation (SAProxies) also under the same threat as PKCs ? Can a perpetrator gain any advantage from generating a bogus token that has the same signature block as an existing one?

Consider the following example [Low94].



Assume X wants to make a request that looks like it is coming from B whilst exercising rights that B has not got e.g. by replacing ar' with enlarged rights, *ar"*. To achieve this X attempts to generate a bogus token, $P'_{B:A}$, that has the same signature block as $P_{B:A}$.

X, however, has several problems:

a. X's key, K^{-}_{X}, is such that when producing the following token

$P'_{B:A}$:

C_B	C_A	$P_{A:S}$	*ar"*	$\text{sig}(P'_{B:A})K^{-}_{X}$

then $\mathbf{sig}(P'_{B:A})\mathbf{K}^{-}_{X} = \mathbf{sig}(P_{B:A})\mathbf{K}^{-}_{A}$.

It is obvious that as this token stands, verification will fail because the public key referenced by C_A which will be used to verify the token signature, is different to K^{+}_{X}, the key needed to verify it.

b. To overcome this X, may replace C_A with C_X in $P'_{B:A}$. Then $P'_{B:A}$ will verify correctly, but the line of authority from $P_{A:S}$ to $P'_{B:A}$ is now broken as $\text{sig}(C_X) \neq \text{sig}(C_A)$ in $P_{A:S}$.

c. X therefore has to find a bogus CA key pair where $K^{-}_{C'}$ is chosen so that $\mathbf{sig}(C_X)\mathbf{K}^{-}_{C'} = \mathbf{sig}(C_A)\mathbf{K}^{-}_{C}$. Then the line of authority will follow from one token to another and the right public keys are pointed at so that the signatures of the tokens of delegation are valid. However, to get this far X has had to introduce a bogus CA key pair and a bogus PKC, thus reducing the problem to that described and dealt with in 2.1.

It is obvious from the above example that any X other than B would gain no benefit in replaying B's request as stated above, as X would only be able to repeat what B is allowed to do anyway. This would effectively be the same as repeating the original request without modification.

Therefore, either B has to be the perpetrator and sign a new request to do more than he was allowed to do, or any other X would have to generate a bogus PKC for B as well with which he can then generate and validly sign a new request.

If X has violated B's CA and generated a bogus PKC that is signed by the correct CA private key, then the verifier may be more easily persuaded that B's new PKC is valid. X then also has to change $\text{sig}(C_B)$ in $P_{B:A}$.

In all cases the perpetrator always has to generate at least one PKC with a new, bogus CA private key and this can be dealt with as in 2.1.

4. Advantages of using Nested Signatures

We have already argued that PKCs should include the CA's public key. We now consider whether SAProxies should also contain public keys and not PKC signature blocks.

If only a public key (and not a PKC) is placed in a token, then the identity of the owner of the public key has to be determined by some other means. There is then no independent 'electronic identity' which can accompany a request so that full verification can be done locally. Authentication of such users then requires communication between the verifier and the CA of the user being authenticated and this is not easily achieved in a distributed system.

It is possible to embed the whole PKC in a SAProxy i.e. include the whole PKC and not just the PKC signature block in the SAProxy and therefore the SAProxy signature. However, a PKC is an unforgeable entity on its own and as such does not need to be covered by another signature. The only requirement is that the identity of the PKC owner is tied to the SAProxy which is achieved by nesting just the signature block.

If the whole PKC is embedded in the SAProxy the amount of data to be verified increases in every SAProxy and efficiency is noticeably reduced. It is then not possible to achieve the significant improvement in performance, both in the amount of data that has to be transmitted as well as verified, that is possible by caching tokens.

Provided that the format of a PKC is modified to include the public key of the CA generating PKCs, the observation made in 1 does not pose a threat to the SAProxy scheme because:

a. it is then computationally infeasible to produce a PKC with a predetermined signature block.

b. without being able to produce such a PKC, verification of nested signature blocks referring to false SAProxies will always fail.

Although this modification results in extra data being present in a PKC, the cost of this is negligible as it only occurs in PKCs and not in all SAProxies. The SAProxy scheme would therefore gain no strength from having public keys or PKCs embedded in the SAProxies and it would be disadvantaged by a loss in performance.

5. Conclusion

We have shown that a CA's public key should be included in the PKCs that it generates. We have then shown that there is no need for any further modifications to certificates or SAProxies to ensure that genuine SAProxies cannot be used with false PKC certificates.

References

[LoCh94a] Low M.R., Christianson B. A Technique for authentication, access control and resource management in open distributed systems. IEE Electronics Letters, 30(2):124-125, January 1994.

[LoCh94b] Low M.R., Christianson B. Self Authenticating Proxies. *Conputer Journal*, 37(5):422-428, October 1994.

[Low94] Low M.R. *Self Defence in Open Systems: Protecting and Sharing Resources in a Distributed Open Environment* Hatfield: University of Hertfordshire, Computer Science Division. Thesis (PhD), September 1994.