Transitive Signature Schemes

Silvio Micali and Ronald L. Rivest

Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, rivest@mit.edu

Abstract. We introduce and provide the first example of a transitive digital signature scheme. Informally, this is a way to digitally sign vertices and edges of a dynamically growing, transitively closed, graph G so as to guarantee the following properties:

- Given the signatures of edges (u, v) and (v, w), anyone can easily derive the digital signature of the edge (u, w).
- It is computationaly hard for any adversary to forge the digital signature of any new vertex or other edge of G, even if he can request the legitimate signer to digitally sign any number of G's vertices and edges of his choice in an adaptive fashion (i.e., even if he can choose which vertices and edges the legitimate signer should sign next after he sees the legitimate signatures of the ones requested so far).

Keywords: public-key cryptography, digital signatures, graphs, transitive closure.

1 Introduction

Sometimes cryptosystems have (or can be designed to have) algebraic properties that make them exceptionally useful for certain applications.

For example, cryptosystems with appropriate homomorphisms can be used for "computing with encrypted data" [14, 8, 9, 15].

Similarly, blind signatures [4, 5] utilize similar homomorphic properties.

Of course, algebraic properties of cryptosystems are often undesirable; they may yield avenues for attacking the cryptosystem such as undesirable "malleability" properties [7].

We propose here a new property for signature schemes that may have interesting applications. We call signature schemes with this property "transitive" signature schemes, because the signature scheme is compatible with computing the transitive closure of the graph being signed.

Subsequent to our work, Johnson et al.[11] have investigated related generalizations under the rubric of "homomorphic signature schemes."

Graphs are commonly used to represent a binary relation on a finite set. A graph $G = (V, E)$ has a finite set V of vertices and a finite set $E \subseteq V \times V$ of edges. We write an edge from u to v as the ordered pair (u, v) in any case, whether the graph is directed or undirected. Cormen et al.[6] describe graph representations and algorithms.

Many graphs are naturally *transitive*: that is, there is an edge from u to v whenever there is a path from u to v.

For a directed example, consider a graph representing a military chain-of-command. Here vertices represent personnel and a directed edge (u, v) from u to v means that u commands (or controls) v. Clearly, if u commands v and v commands w, then u commands w.

For an undirected example, consider a graph representing a set of administrative domains. The vertices represent computers and an undirected edge means that u and v are in the same administrative domain. Again, it is clear that if u and v are in the same administrative domain, and if v and w are in the same administrative domain, then u and w are in the same administrative domain. Transitive (and reflexive) undirected graphs represent equivalence relations.

We are interested in situations where someone (let's call her Alice) wishes to publish a transitive graph in an authenticated (i.e., signed) manner. Signing the graph allows others to know that they are working with the authentic graph (or with authentic components of the graph).

Of course, Alice could just sign a representation of the entire graph as a single signed message. This approach may, however, be awkward in practice, particularly if the graph changes frequently or the components are large.

Regarding the efficiency of representation, we observe that a graph with n vertices may have $O(n^2)$ edges. It may be more efficient for Alice to sign a smaller graph, with the explicit understanding that the intended graph is the *transitive closure* of the signed graph. In this way she never needs to sign more than $O(n)$ edges (in the undirected case).

The *transitive closure* $G^* = (V^*, E^*)$ of a graph $G = (V, E)$ is defined to have $V^* = V$ and to have an edge (u, v) in E^* if and only if there is a path (of length zero or greater) from u to v in G. (This is more properly called the reflexive transitive closure, but we stick with standard usage.)

For further efficiency, we now restrict our attention to schemes wherein Alice signs the vertices and edges of the graph G individually. This allows G to grow dynamically: vertices and edges may be added later on without Alice having to re-sign everything done before. (We assume that vertices and edges are never deleted.)

Alice thus has two signature schemes for signing the components of the graph G: one signature scheme for signing vertices, and one for signing edges. We denote her signature of the vertice v as $\sigma(v)$ and her signature of the edge (u, v) as $\sigma(u, v)$, with the understanding that the underlying signature schemes may be different.

Because we are focussing on the situation where the graph Alice intends to sign is transitive, she need only sign a subset of the edges, as long as the transitive closure of that subset is equal to the intended graph. For example, in the chain-of-command example, she need only sign edges representing the relationship between an individual and his immediate superior; other edges can be inferred from these. Or, for another example, in the administrative domain

example, she need only sign enough edges to form a spanning tree within each administrative domain.

An observer who then sees a signed edge $\sigma(u, v)$ and another signed edge $\sigma(v, w)$ can then infer that (u, w) is also in the graph being signed by Alice. It is as if Alice had actually signed the edge (u, w) directly. And then if the observer sees another edge (w, x), he can infer that (u, x) is in the graph as well.

In some applications, however, it may be necessary for a party to actually *prove* that Alice signed (either explicitly or implicitly by transitivity) an edge. For example, a party x may need to prove that he is in the chain of command underneath party u. Or, party x may need to prove that he is in the same administrative domain as party u. How can this be done, if the edge (u, x) was not explicitly signed, but is only inferrable by transitivity?

A straightforward approach to proving that (u, x) is in the graph is to produce a "proof" consisting of a sequence of signed edges forming a path from u to x (with their signatures). In this example, a proof that (u, x) is in the graph might consist of the sequence:

$$(u, v), (v, w), (w, x) \tag{1}$$

and their corresponding signatures:

$$\sigma(u, v), \sigma(v, w), \sigma(w, x) \ . \tag{2}$$

(Actually, the proof includes Alice's signatures on each of the vertices as well:

$$\sigma(u), \sigma(v), \sigma(w), \sigma(x) \ .) \tag{3}$$

This sequence of signed edges forms a path from u to x, thus proving that the edge (u, x) is in the graph being signed by Alice.

The problem with using such "path-proofs" is that they may become cumbersome if the path is long, and may introduce unnecessary detail and information in the proof. (Why should a soldier need to mention each of his superior officers if all he is trying to prove is that he is in the U.S. Army?)

We are thus led to wonder whether some signature schemes might be compatible with a "path compression" operator that produces an inferred signature that is indistinguishable from one that might have been produced by Alice.

More specifically, let us informally define a *"transitive signature scheme" to be a scheme for signing the vertices and edges of a graph such that if someone sees Alice's signatures on vertices u, v, and w and also sees Alice's signatures on edges (u, v) and (v, w), then that someone can easily compute a valid signature on the edge (u, w) that is indistinguishable from a signature on that edge that Alice would have produced herself.* See Figure 1.

With a transitive signature scheme Alice only needs to sign a minimum number of edges that has the same transitive closure as her intended graph; an observer can infer her signatures on the remaining (inferred) edges.

We note for the record that this minimum subset of edges having the same transitive closure is called the *transitive reduction* of a graph and can be computed efficiently in both the undirected and directed cases[1].

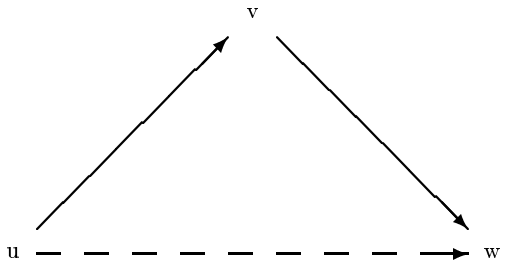

Fig. 1. With a transitive signature scheme anyone can compute a signature on edge (u, w) given signatures on edges (u, v) and (v, w).

With a transitive signature scheme, anyone can produce a short "proof" that a given edge is in the graph. Even if Alice didn't sign that edge explicitly, the proof is a signature that might as well been produced by Alice. Most conveniently, one does not need to have the verifier understand chains or sequences of signed edges (as one typically has to do for certificate chains in an analagous situation, for example[12, Section 13.6.2]).

Having providing some motivation, we can then ask: do transitive signature schemes exist?

We provide a partial answer below.

2 An undirected transitive signature scheme

In this section we describe a transitive signature scheme for working on undirected graphs (which we dub a *undirected transitive signature scheme*) and prove it secure. It is based on the difficulty of the discrete logarithm problem.

User setup: Each user selects a public-key signature scheme for signing vertices. The vertex signature scheme should have the usual security properties, but need not have any special algebraic properties. For example, the scheme proposed by Goldwasser, Micali, and Rivest[10] is satisfactory here as a vertex signature scheme. The user selects a public-key/private-key pair, and publishes the public-key.

For use in signing edges, each user selects and publishes the following parameters:

- large primes p and q such that q divides $p - 1$, and
- generators g and h of the subgroup G_q of order q of $\mathbf{Z}_p^*$, such that the base-g logarithm of h modulo p is infeasible for others to compute.

Creating a new vertex: When a user Alice wishes to create a new vertex and add it to the graph, she does the following:

- Let n denote the number of vertices previously created. Increment n by 1.
- Select two values x_n and y_n independently at random from $\mathbf{Z}_q$.
- Compute v_n as $g^{x_n} h^{y_n} \pmod{p}$.
- Compose, sign, and publish a statement of the form, "The n-th vertex of the graph is represented by the value v_n." (The message is signed using the vertex-signing public-key signature scheme. The value v_n is given explicitly in the message. The values x_n and y_n are kept secret by Alice.)

Signing the edge (i, j): To sign the edge between the i-th vertex and the j-th vertex, Alice computes and publishes the quadruple:

$$(i, j, \alpha_{ij}, \beta_{ij})$$

where

$$\begin{aligned} \alpha_{ij} &= x_i - x_j \pmod{q} \\ \beta_{ij} &= y_i - y_j \pmod{q} . \end{aligned}$$

We note that the vertex-signing procedure is very similar to the information-theoretically secure commitment scheme of Pedersen[13] (or equivalently, of Chaum, van Heijst, and Pfitzmann[3]).

Verifying an edge signature: Anyone can verify the signature on an edge by checking that

$$v_i = v_j g^{\alpha_{ij}} h^{\beta_{ij}} \pmod{q} . \tag{4}$$

Composing edge signatures: Given a signature $(i, j, \alpha_{ij}, \beta_{ij})$ of edge (i, j) and a signature $(j, k, \alpha_{jk}, \beta_{jk})$ of edge (j, k), anyone can compute the signature

$$(i, k, \alpha_{ik}, \beta_{ik})$$

on edge (i, k) as:

$$\begin{aligned} \alpha_{ik} &= \alpha_{ij} - \alpha_{jk} \pmod{q} \\ &= x_i - x_k \pmod{q} \\ \beta_{ik} &= \beta_{ik} - \beta_{ik} \\ &= y_i - y_k \pmod{q} . \end{aligned}$$

This signature is identical to what Alice would produce when signing the edge (i, k).

3 Security

As with any signature scheme, proving security means proving that an adversary will not be able to forge new signatures having seen some previous legitimate signatures. Of course, when the signature scheme has the kind of algebraic property considered in this paper, the adversary is intentionally given for free the

ability to "forge" signatures on new edges, as long as they are in the transitive closure of previously signed edges. Since the adversary is being explicitly given this capability for free, it is considered a feature (and not a defect) that the adversary can compute these signatures. It is thus the ability of the adversary to compute signatures on edges *outside* the transitive closure of previously signed edges that should be considered as "forgery" and a break of the signature scheme; our proof shows that forging signatures outside of the transitive closure of previously signed edges is provably hard.

There is another subtle issue regarding the definition of security, regarding how the previously signed edges are determined. The simplest sort of "static" adversary would be asked to forge a signature on a "new" edge, given as input a graph with a given set of edge signatures.

A more powerful adversary, which our scheme is capable of defeating, would be given the following capabilities, which he could exercise at will, until he is ready to attempt a forgery. The adversary can make the following requests of the signer.

- **Initialize.** Ask the signer to discard any previously signed vertices and edges, and begin with a "clean slate"; ask the signer to make up a new name N for a new graph to be constructed.
- **Add a new vertex i.** Ask the signer to sign a statement saying that "vertex i is now part of the graph N."
- **Add a new edge (i, j).** Ask the signer to sign a statement saying that "edge (i, j) is now part of the graph N."

Once the adversary is done with his requests, he is challenged to forge an edge signature on any edge of his choice, as long as the edge is not in the transitive closure of the edges previously signed.

It is important to emphasize that the adversary is *adaptive* in the sense that each such request need be formulated by the adversary only after he has seen the response to all previous requests. The adversary does not need to commit to all of his requests in advance; he makes them up as he goes along.

(We note that if the adversary is static rather than adaptive, in the sense that he must commit to *all* of requests before seeing the responses to any of them, then other simpler schemes will work. For example, it suffices for the signer to publicly assign a random number x_i to each vertex i, and to sign edge (i, j) by giving an RSA signature on the value x_i/x_j, that is, $(x_i/x_j)^d \pmod n$. The multiplicative property of RSA ensure the desired transitive property. However, we don't know how to prove this scheme is secure against an adaptive adversary, as the usual reduction techniques seem to require knowing ahead of time (when the x_i values are committed to) what the final connected components will be, so that the x_i values can be appropriately set up for the reduction, and there are too many possible arrangements of the connected components to guess it correctly with a inverse polynomial chance of success.)

In our scheme, starting a new graph means making up a new set of system parameters for that graph, and publishing the details of the corresponding vertex

and edge signature schemes. Each vertex and edge signature is accomplished as described in the previous section.

Theorem 1. *The proposed undirected transitive signature scheme is secure in the sense that an adversary can not forge a signature for an edge not in the transitive closure of edges already seen, even if he can adaptively request new vertices to be created and signed and request edges to be signed, assuming that computing discrete logarithms is hard.*

Proof sketch: A straightforward reduction from the discrete logarithm in G_q, the subgroup of prime order q modulo p. Given an instance (g, y, p, q) of the discrete logarithm problem (where g and y are in G_q and the goal is to compute $\log_g(y) \bmod p$) we create a transitive signature scheme with $g = g$ and $h = y$. It is easy to see how a simulator can respond successfully to requests for signatures of edges. However, the usual linear algebra (see Pedersen[13]) can be used to show that it is hard for an adversary to forge signatures since it would imply being able to compute $\log_g(h)$. One needs to argue that the adversary learns nothing about the true x_i and y_i values from the signatures he observes. The adversary's forgery on an edge is thus extremely unlikely to be equal to the simulator's signature for that edge. Given two signatures on the same edge, $\log_g(h)$ can be computed. (A more complete proof will be given in the complete version of this paper.) □

4 Remarks and Discussion

The problem of finding a *directed* transitive signature scheme remains a very interesting open problem. We have not been able to make much progress on this problem.

We note that there is some similarity between our problem and the problem of "atomic proxy cryptography" due to Blaze et al.[2]. However, we have not been able to come up with a provably secure undirected transitive signature scheme for a scheme modelled on their ideas without losing too much efficiency in the proof.

Another interesting open problem is the following: Given Alice's signature on message M and on (in some special way) Bob's public key, Bob can then "cut himself out of the middle" and produce Alice's signature on M and on (in the same special way) Carol's public key. This would be very useful for collapsing chains of delegation or certification.

Acknowledgments

We would like to thank Shafi Goldwasser for Mihir Bellare for many interesting discussions, encouragement, and suggestions. (In particular, Mihir Bellare sent us a proof that the RSA-based scheme give works against a static adversary.)

References

1. A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1:131–137, 1972.
2. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Proceedings EUROCRYPT '98*, pages 127–144. Springer, 1998.
3. D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In J. Feigenbaum, editor, *Proceedings CRYPTO '91*, pages 470–484. Springer, 1992. Lecture Notes in Computer Science No. 576.
4. David Chaum. Blind signatures for untraceable payments. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proceedings CRYPTO 82*, pages 199–203, New York, 1983. Plenum Press.
5. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, Oct 1985.
6. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.
7. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proc. STOC '91*, pages 542–552. ACM, 1991.
8. Joan Feigenbaum. Encrypting problem instances: Or...can you take advantage of someone without having to trust him? In H. C. Williams, editor, *Proceedings CRYPTO 85*, pages 477–488. Springer, 1986. Lecture Notes in Computer Science No. 218.
9. Joan Feigenbaum and Michael Merritt. Open questions, talk abstracts, and summary of discussions. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 2, pages 1–45, 1991.
10. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM*, 17(2):281–308, April 1988.
11. Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *Proceedings RSA 2002*, 2002.
12. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
13. T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Proceedings CRYPTO '91*, pages 129–140. Springer, 1992. Lecture Notes in Computer Science No. 576.
14. Ronald L. Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 169–180. Academic Press, 1978.
15. Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *Proceedings 40th IEEE Symposium on Foundations of Computer Science*, pages 554–566, New York, 1999. IEEE.