Pairings and Trusted Computing

Nigel Smart

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB.

Joint work with L. Chen and P. Morrissey.

August 31, 2008

Introduction

Pairings are often considered inefficient compared to standard primitives.

Pairings are often considered to only be useful in non-standard constructions (IBE, etc)

We will show that pairings can be used to make protocols much more efficient.

Our example will be the Direct Anonymous Attestation protocol which forms part of the the Trusted Computing Group standard.

- This must be one of the most complex cryptographic protocol currently deployed.

Symmetric Pairings

We start by considering a symmetric pairing

$$\hat{t} : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_T$$

between groups $\mathbb{G}_1 = \langle P \rangle$ of prime order q

We all know the problems with symmetric pairings

- Security does not scale well, $k \leq 6$

However the main benefit is that we can solve DDH in $\mathbb{G}_1$ using symmetric pairings by testing

$$\hat{t}(aP, bP) = \hat{t}(P, cP).$$

This does not hold in general for asymmetric pairings,

- i.e. it is not true that pairings always provide a DDH oracle.

Camenisch-Lysyanskaya Signature Scheme

KeyGeneration:

The private key is a pair $(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q$.
The public key is given by the pair

$$X = [x]P \text{ and } Y = [y]P$$

Signing:

On input of a message $m \in \mathbb{Z}_q$ the signer generates $A \in \mathbb{G}_1$ at random and outputs the (A, B, C) where

$$B = [y]A \text{ and } C = [x + mxy]A.$$

Verification:

The verifier checks whether

$$\hat{t}(A, Y) = \hat{t}(B, P) \text{ and } \hat{t}(A, X) \cdot \hat{t}([m]B, X) = \hat{t}(C, P).$$

Camenisch-Lysyanskaya Signature Scheme

Verification works since

$$\hat{t}(A, Y) = \hat{t}(A, [y]P) = \hat{t}([y]A, P) = \hat{t}(B, P)$$

and

$$\hat{t}(A, X) \cdot \hat{t}([m]B, X) = \hat{t}(A + [my]A, xP) = \hat{t}([x + mxy]A, P).$$

Re-Randomization of Signatures

Interestingly the Camenisch-Lysyanskaya signature scheme allows re-randomization of the signatures.

Given a signature (A, B, C) and a random element $r \in \mathbb{Z}_q$ then (rA, rB, rC) is also a valid signature.

Using the DDH oracle provided by the symmetric pairing

- One can tell whether a signature (A', B', C') corresponds to a randomization of another signature (A, B, C).
- If it did we would have $A' = rA$, $B' = rB$ and $C' = rC$ for some value r, and so

$$\hat{t}(A', B) = \hat{t}(A, B') \text{ and } \hat{t}(A', C) = \hat{t}(A, C').$$

Re-Randomization of Signatures

But if you wanted to re-randomize signatures to avoid linkage between the two (without knowing the message) then the above is exactly what we do not want.

- In DAA we want a single signature from the Issuer to be associated with many transactions, but each transaction should not be linked.

It is the DDH oracle which is the problem.

We can get around this by adopting a more complicated protocol.

Proof of Knowledge of Signatures

Given a signature (A, B, C) we can create a proof of knowledge of this signature via

- r, r' chosen at random
- $A' = [r']A$, $B' = [r']B$, $C' = [r \cdot r']C$.
- Send A', B', C' to verifier
- Both parties compute

$$v_x = \hat{t}(A', X), \quad v_{xy} = \hat{t}(B', X), \quad v_s = \hat{t}(C', P)$$

- Prover proves knowledge of $(u, v) = (1/r, m)$ such that

$$v_s^u = \hat{t}(C, P)^{r'} = \hat{t}(A, X)^{r'} \cdot \hat{t}(mB, X)^{r'} = v_x \cdot v_{xy}^v$$

- Verifier checks that $\hat{t}(A', Y) = \hat{t}(B', P)$.

Note, need additional randomization of C' and complicated proof to avoid linkage.

Proof of Knowledge of Signatures

To get around the complexity of the above re the randomization we could use asymmetric pairings.

$$\hat{t} : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$$

between groups $\mathbb{G}_1 = \langle P_1 \rangle$ and $\mathbb{G}_2 = \langle P_2 \rangle$ of prime order q

- Should also not have efficient homomorphisms from $\mathbb{G}_2$ to $\mathbb{G}_1$ (or vice-versa).

This leads to an asymmetric version of the Camendisch-Lysyanskaya signature.

Camenisch-Lysyanskaya Signature Scheme

KeyGeneration:

The private key is a pair $(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q$.
The public key is given by the pair

$$X = [x]P_2 \text{ and } Y = [y]P_2$$

Signing:

On input of a message $m \in \mathbb{Z}_q$ the signer generates $A \in \mathbb{G}_1$ at random and outputs the (A, B, C) where

$$B = [y]A \text{ and } C = [x + mxy]A.$$

Verification:

The verifier checks whether

$$\hat{t}(A, Y) = \hat{t}(B, P_2) \text{ and } \hat{t}(A, X) \cdot \hat{t}([m]B, X) = \hat{t}(C, P_2).$$

Re-Randomization of Signatures

Can now re-randomize via

$$A' = rA, \;\; B' = rB, \;\; C' = rC$$

Do not have linkage due all elements being in $\mathbb{G}_1$ and no DDH oracle.

A similar but simplified proof of knowledge performed to prove knowledge of the original signature.

The DAA Protocol

The DAA protocol is for one machine to attest to another machine that it has a given configuration.

However, this attestation needs to be done in a way which preserves the anonymity of the first machine.

Yet the first machine must be able to choose whether it wishes different transactions to be linked.

DAA Example : 1

Suppose you arrive in a Uni and want to print some important confidential document.

You send this to the printer, but how do you know that the printer does not send the document to someone else, as well as printing it ?

You need to know that the printer has not been nobbled

You want the printer to attest to you that it has a particular configuration

So somewhere in the printer must be a piece of hardware that you trust

- This is called the Trusted Platform Module (or TPM)

DAA Example : 2

Now consider an online shopping scenario.

You want to buy some software from a web site, e.g. an OS upgrade.

The web site will only supply this if you have a genuine original OS on your system.

So your computer attests to the merchant that your OS is valid.

But you do not want your identity revealed.

Next you buy a computer game.

You would like the two transactions linked

- Maybe because you earn some loyalty points or something
- But despite this linking, you still want to maintain your anonymity

DAA Example : 2

What happens if someone spots you are doing something wrong

- Maybe your TPM has been hacked or something

DAA assumes that bad guys are nice enough to publish all keys of TPM's they compromise.

- This is nuts, but thats DAA for you

The good guys should be able to work out whether a signature has been produced by a hacked TPM

- This is termed Rogue Tagging.
- Requires the bad guy helps by publishing the key he has compromised.

The DAA Players

There are three types of players, users, issuers and verifiers.

A set of users $\mathcal{U}$ where each $U_i \in \mathcal{U}$ consists of

- A TPM m_i from some set of TPMs $\mathcal{M}$. Each TPM has
  - An endorsement key ek_i
  - A seed string $\mathsf{DaaSeed}_i$
- A Host h_i from some set of host. Each Host has
  - A value cnt_i.
  - A set of commitments $\{\mathsf{comm}\}_i$
  - A set of credentials $\{\mathsf{cre}\}_i$.

The TPM generates a secret f from the $\mathsf{DaaSeed}_i$.

A credential is essentially a signature of knowledge of an f/commitment pair.

The DAA Players

A set of issuers $\mathcal{I}$ where each $I_k \in \mathcal{I}$ has

- A public and private key pair $(\mathrm{ipk}_k, \mathrm{isk}_k)$.
- A long term value K_k (e.g. the long term public key of the issuer).
- It also maintains a list of rogue TPM internal values, we denote this list by $\mathrm{RogueList}(I_k)$.

A set of verifiers $\mathcal{V}$, where each verifier $V_j \in \mathcal{V}$ maintains

- A set of base names $\{\mathrm{bsn}\}_j$
- A list of rogue TPM internal values $\mathrm{RogueList}(V_j)$.
- Each V_j may optionally maintain a list of message and signature pairs received (this can be used to trade memory for computation in linking).

A DAA Scheme

This is a tuple of protocols and algorithms

$$\mathsf{Daa} = (\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Link}, \mathsf{RogueTag})$$

where:

Setup(1^t) is a p.p.t. system setup algorithm.
This outputs a set of system paramters par which contains all of the issuer public keys ipk_k and the various parameter spaces.
It also serves to setup and securely distribute each of the issuer secret keys isk_k.

Join(U_i, I_k) is a 3 party protocol run between a TPM, a Host and an issuer.
The Host obtains from the issuer, with the help of the TPM, a valid credential.

A DAA Scheme

Sign(U_i, msg) is a 2 party protocol run between a TPM and a Host. A signature on a message wrt knowledge of a credential is created, subject to a basename, which may (or may not) allow the signature to be linked to other signatures.

- Signatures with the same basename should be linkable.

Verify(σ, msg) is a d.p.t. verification algorithm that allows a given verifier to verify a signature of knowledge σ of a credential on a message msg wrt a basename.
The verification process will involve checking the signature against the list RogueList(V_j).

A DAA Scheme

Link(σ_0, σ_1) is a $\mathrm{d.p.t.}$ linking algorithm that returns either *linked*, *unlinked* or $\perp$.
The algorithm should return

- $\perp$ if either signature was produced with a rogue key,
- *linked* if both are valid signatures and the user who produced them wanted these to be linkable to each other,
- *unlinked* otherwise.

RogueTag(f, σ) is a $\mathrm{d.p.t.}$ rogue tagging algorithm.
This returns true if σ is a valid signature produced using the TPM secret value f and returns false otherwise.

A DAA Scheme

For correctness we require that if

- A user $U_i \in \mathcal{U}$ engages in a run of Join with I_k, resulting in U_i obtaining a commitment comm on a TPM secret value f and a credential cre corresponding to f,
- and the user U_i then creates two signatures σ_b on two messages msg_b for $b \in \{0, 1\}$ intended for verifier $V_j \in \mathcal{V}$ with basename bsn (which could be $\perp$),
- and the secret TPM value used to compute these f is not in RogueList.

Then

$$\mathsf{Verify}(\sigma_0, \mathsf{msg}_0) = \mathsf{Verify}(\sigma_1, \mathsf{msg}_1) = \textit{accept}$$

and if bsn $\neq \perp$ then $\mathsf{Link}(\sigma_0, \sigma_1) =$ *linked*.

Factoring Based DAA Scheme

The current standardised DAA scheme is based on another signature scheme of Camenisch and Lysyanskaya based on factoring.

Let N be the RSA modulus.

This defines an RSA group $\mathbb{Z}_N$.

In addition there is a finite field $\mathbb{F}_\Gamma$ in which the DLP is hard.

The cost of an exponentiation operation in these two groups we denote by $\mathbb{G}_N$ and $\mathbb{G}_\Gamma$.

- $\mathbb{G}_N^3$ is the cost of a multi-exponentiation with three terms.

Factoring Based DAA Scheme

Operation	Party	Cost
Join	TPM	$3 \cdot \mathbb{G}_\Gamma + 2 \cdot \mathbb{G}_N^3$
	Issuer	$n \cdot \mathbb{G}_\Gamma + 2 \cdot \mathbb{G}_N + 1 \cdot \mathbb{G}_N^4 + 1 \cdot {\mathbb{G}_\Gamma}^2 + P_c$
	Host	$1 \cdot \mathbb{G}_\Gamma + 1 \cdot \mathbb{G}_N^2 + P_v$
Sign	TPM	$3 \cdot \mathbb{G}_\Gamma + 1 \cdot \mathbb{G}_N^3$
	Host	$1 \cdot \mathbb{G}_\Gamma + 1 \cdot \mathbb{G}_N + 1 \cdot \mathbb{G}_N^2 + 2 \cdot \mathbb{G}_N^3 + 1 \cdot \mathbb{G}_N^4$
Verify	Verifier	$4 \cdot {\mathbb{G}_\Gamma}^2 + 2 \cdot \mathbb{G}_N^4 + 1 \cdot \mathbb{G}_N^6 + n\mathbb{G}_\Gamma$
RogueTag	Verifier	$1 \cdot \mathbb{G}_N^4$

- P_v is the cost of generating a prime number and P_c is the cost of verifying a prime number
- n is the number of keys in the verifiers rogue list

The TPM is a small constrained device, yet we can see it has to perform a lot of cryptographically complicated operations.

Symmetric Pairing Based DAA Scheme

To deal with this problem Brickell, Chen and Li presented a symmetric pairing based scheme.

Operation	Party	Cost
Join	TPM	$3 \cdot \mathbb{G}_1$
	Issuer	$(2+n) \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_1^2$
	Host	$6 \cdot P$
Sign	TPM	$4 \cdot \mathbb{G}_T$
	Host	$3 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_T + 3 \cdot P$
Verify	Verifier	$(n+1) \cdot \mathbb{G}_T + 1 \cdot \mathbb{G}_T^2 + 1 \cdot \mathbb{G}_T^3 + 5 \cdot P$
RogueTag	Verifier	$1 \cdot \mathbb{G}_T$

Notice the TPM in joining needs to do a simple operation.

On signing however the TPM has to do a lot of work since operations in $\mathbb{G}_T$ are not simple.

Symmetric Pairing Based DAA Scheme

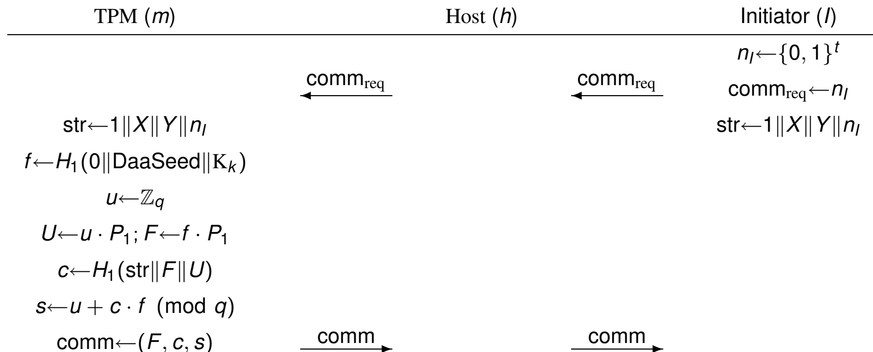
The protocol is based on the pairing based Camenisch-Lysyanskaya signature scheme in the symmetric pairing setting.

Due to the DDH oracle a lot of work needs to be done to avoid linking signatures.

By moving the protocol to the asymmetric pairing setting one avoids this problem.

- A number of other optimizations then become apparent.

Join Protocol - Part I



Join Protocol - Part II

TPM (m)		Host (h)		Initiator (I)
				$U' \leftarrow sP_1 - cF$
				If $F = f \cdot P_1$ for some f on the rogue list, or $c \neq H_1(\text{str} \| F \| U')$ then **abort**
				$r \leftarrow \mathbb{Z}_q$
				$A \leftarrow r \cdot P_1$; $B \leftarrow y \cdot A$
				$C \leftarrow (x \cdot A + rxy \cdot F)$
				$\text{cre} \leftarrow (A, B, C)$
	$\xleftarrow{\mathcal{E}}$		$\xleftarrow{\mathcal{E}}$	$\mathcal{E} \leftarrow E_{\text{ek}}(\text{cre})$

Join Protocol - Part III

TPM (m)		Host (h)	Initiator (I)
$\mathrm{cre} \leftarrow E_{\mathrm{ek}}^{-1}(\mathcal{E})$; $E \leftarrow f \cdot B$	$\xrightarrow{\mathrm{cre},\, E}$	$\rho_a \leftarrow \hat{t}(A, X)$	
		$\rho_b \leftarrow \hat{t}(B, X)$	
		$\rho_c \leftarrow \hat{t}(C, P_2)$	
		If $\hat{t}(A, Y) \neq \hat{t}(B, P_2)$	
		or $\hat{t}(A + E, X) \neq \rho_c$	
		then **abort**	

Join Protocol Notes

The value of cre is not sent in the clear and hence only the intended user can obtain the complete credential.

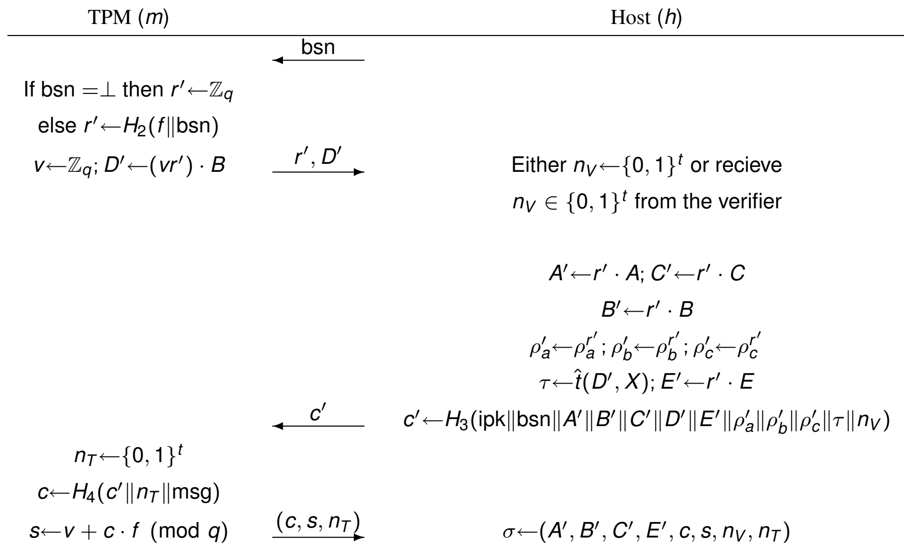
In contrast with the RSA-based DAA schemes we do not require a complicated proof of knowledge of the correctness of a given commitment.

- Instead, the proof of knowledge is provided by a very efficient Schnorr signature, on the value F computed using the secret key f.

Once a credential is issued from I, the TPM and the Host verify that this credential is correctly formed.

The values ρ_a, ρ_b, ρ_c and E are stored for later use by the Host in the signing algorithm.

Sign Protocol



Sign Protocol Notes

In most applications of the Sign protocol, the signature generated as a request from the verifier, and the verifier supplies its own value of n_V, to protect against replays of previously requested signatures.

The value bsn is used to link signatures. Unless bsn $=\perp$.

The use of E' allows the verifier to identify if the signature was produced by a rogue TPM by computing $[f_i]B'$ for all f_i values on the rogue list and comparing these to E'.

The value r' is used to mask the signature created from the other players in the scheme including the issuer.

Verification Algorithm

On input a signature σ of the form

$$\sigma = (A', B', C', E', c, s, n_V, n_T)$$

this algorithm performs the following steps:

1. Check Against Rogue List. If $E' = B'^{f_i}$ for any f_i in the set of rogue secret keys then return *reject*.
2. Check Correctness of A' and B'. If $\hat{t}(A', Y) \neq \hat{t}(B', P_2)$ then return *reject*.
3. Verify Correctness of Proofs. This is done by performing the following sets of computations:
   - $\rho_a^\dagger \leftarrow \hat{t}(A', X)$, $\rho_b^\dagger \leftarrow \hat{t}(B', X)$ and $\rho_c^\dagger \leftarrow \hat{t}(C', P_2)$.
   - $\tau^\dagger \leftarrow (\rho_b^\dagger)^s \cdot (\rho_c^\dagger / \rho_a^\dagger)^{-c}$
   - $D^\dagger \leftarrow sB' - cE'$.
   - $c^\dagger \leftarrow H_3(\mathrm{ipk} \| \mathrm{bsn} \| A' \| B' \| C' \| D^\dagger \| E' \| \rho_a^\dagger \| \rho_b^\dagger \| \rho_c^\dagger \| \tau^\dagger \| n_V)$.

   Finally if $c \neq H_4\left(c^\dagger \| n_T \| \mathrm{msg}\right)$ return *reject* and otherwise return *accept*.

Linking Algorithm

On input a pair of signatures σ_b for $b \in \{0, 1\}$ each having the form

$$\sigma_b = (A'_b, B'_b, C'_b, E'_b, c_b, s_b, n_{V,b}, n_{T,b})$$

the algorithm performs the following steps:

1. Verify Both Signatures. For each signature σ_b the verifier runs Verify(σ_b) and if either of these returns *reject* then the value $\perp$ is returned.
2. Compare Signatures and Basenames. If the two basenames which verify the signatures are equal, and if $A'_0 = A'_1$ then return *linked*, else return *unlinked*.

Cost of our DAA protocol

Operation	Party	Cost
Join	TPM	$3 \cdot \mathbb{G}_1$
	Issuer	$(2+n) \cdot \mathbb{G}_1 + 2 \cdot \mathbb{G}_1^2$
	Host	$6 \cdot P$
Sign	TPM	$1 \cdot \mathbb{G}_1$
	Host	$4 \cdot \mathbb{G}_1 + 3 \cdot \mathbb{G}_T + 1 \cdot P$
Verify	Verifier	$n\mathbb{G}_1 + 1 \cdot \mathbb{G}_1^2 + 1 \cdot \mathbb{G}_T^2 + 5 \cdot P$
RogueTag	Verifier	$1 \cdot \mathbb{G}_1$

The main point to note is that the TPM is only required to perform operations in $\mathbb{G}_1$, which can be an elliptic curve over a relatively small finite field.

- Especially in the signing phase

The TPM does not have to perform any expensive operations at all.

Summary

By moving to the asymmetric pairing situation and careful analysis of the protocol

- We have a simple DAA protocol
- Removed the need for complex ZK proofs in the factoring version
- Reduced computations of the TPM.