Privacy and Security Issues in e-Ticketing
Optimisation of Smart Card-based Attribute-proving

Jaap-Henk Hoepman[1,2], Bart Jacobs[1], and Pim Vullers[1⋆]

[1] Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.
{jhh, bart, p.vullers}@cs.ru.nl
[2] TNO Information and Communication Technology, The Netherlands.
jaap-henk.hoepman@tno.nl

Abstract This short note concentrates on an optimisation of the attribute-proving protocol by Batina et al. [1], and provides the improved performance figures. The protocol relies on elliptic curve cryptography with bilinear pairings. These pairings provide signatures that are stable under multiplication with a blinding factor. In this way multiple proofs are unlinkable, and thus provides a privacy-friendly solution.
The optimisation involves better exploitation of the (limited) elliptic curve primitives that are available on the current generation of Java Card smart cards. It leads to a reduction of the on-card running times (wrt. to [1]) of roughly a factor three. Total running times with this new protocol are below one second. A further reduction with a factor two or three is needed to achieve performance that is acceptable in practice.

Key words: anonymous credentials, elliptic curve cryptography, smart card, bilinear pairing, attributes, blinding, protocols, Java Card

1 Introduction

With e-ticketing, smart cards replace the use of paper tickets to prove the right of the bearer to use the public transportation system.

Smart cards that are currently employed for e-ticketing in public transport are typically memory cards with cryptographically protected access for reading and writing. Cards have unique identifiers and most of the "intelligence" of the whole system lies in the back office. Now that the cryptographic protection of the most widely used smart card for e-ticketing, the MIFARE Classic, is broken [2,3], this back office plays a crucial role in fraud detection. Fraudulent cards can be recognised using a shadow bookkeeping, and their use can be blocked on the basis of their card-id.

This back office database with extensive logs of the movements of individual cards within the system, often linkable to individuals, is not only used for fraud detection but also for capacity optimisation, division of revenues between

⋆ Sponsored by Trans Link Systems/Open Ticketing.

different transport companies, law enforcement and direct marketing. With the growing awareness of the privacy issues involved, the interest in more privacy-friendly alternatives increases.

Anonymous credentials are the obvious privacy-friendly technique to use. They allow travellers to use a smart card for e-ticketing in a way that is fairly similar to paper tickets, namely as a way of proving access rights to public transport, without revealing one's identity. The main problem with anonymous credentials in this context is that they involve computationally intensive protocols. They require processor-based, instead of memory-based, smart cards, which are generally more expensive. But even with the latest generation of processor cards, processing speed is a mayor challenge. Typically in the mass transit sector the transaction times should be below 300-400 milliseconds, in order to prevent queues at entry/exit gates. In contrast, similar processor cards that are currently used in e-passports don't have such tight constraints. These e-passport protocols easily take a few seconds to complete.

2 Elliptic Curve Background

For the broader context and more background information the reader is referred to the original paper [1]. Here we only describe the basics about the underlying elliptic curve (EC) primitives such that the notation in the optimised protocol in Section 3.2 can be understood.

We shall use $e(-,-)$ to denote a bilinear pairing on an elliptic curve. A private key s is just a natural number (below some bound). A public key associated with the private key s is the result of a scalar point multiplication $s \cdot Q$, for some fixed point Q. Suppose a card c has a public key P_c. A signature on this key, for instance corresponding to an attribute, created by multiplication with the secret key, is thus a point $s \cdot P_c$. It may be given to the card upon initialisation.

The card c can now prove that it possesses a signed public key by showing a point R, which is claimed to be $s \cdot P_c$. This can be verified on the terminal side by checking the equality:

$$e(P_c,\ s \cdot Q) \stackrel{?}{=} e(R,\ Q).$$

If $R = s \cdot P_c$, then by bilinearity of e both sides are equal to $e(P_c,\ Q)^s$.

One of the attractions of this kind of signature is that it is stable under blinding. Instead of the pair $(P_c, s \cdot P_c)$ the card can also present $(b \cdot P_c, b \cdot (s \cdot P_c))$, using that $b \cdot (s \cdot P_c) = s \cdot (b \cdot P_c)$, where b is the blinding factor. Hence the card can present different looking credentials each time, making tracing impossible [4].

What the terminal needs to check in this scenario is that P_c really is a public key of the card c. It can check using some standard challenge-response mechanism that c possesses the corresponding private key. This is what is done in [1], using some separate protocol steps, see Section 3.1.

The optimisation that is presented in Section 3.2 involves an integration of a different challenge-response mechanism in the attribute verification steps. This

makes the protocol shorter and more efficient, especially because it has fewer messages and can be implemented entirely using calls that are part of the Java card API. The earlier version in [1] had to rely on our own (slow) multiplication algorithm for blinding, implemented in Java Card on the card.

3 Protocols for Attribute-proving

This section describes the optimised version of the elementary protocol designed by Batina *et al* [1]. We describe how a card c demonstrates in a secure and privacy friendly manner that it possesses some attribute(s) a. This attribute is just a number, with certain meaning that we abstract away.

3.1 Sketch of the Original Protocol

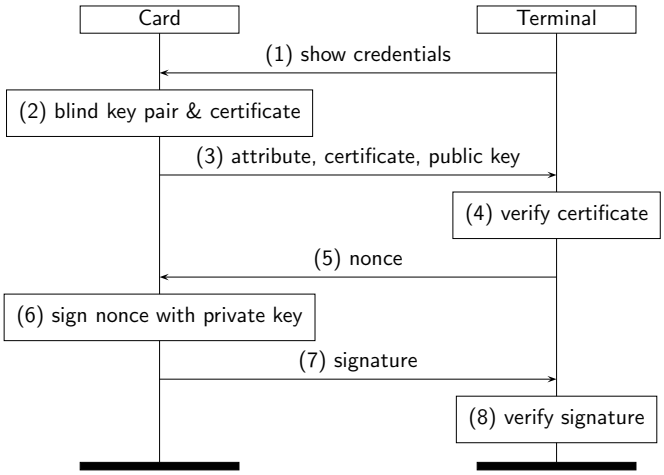

Figure 1. Sketch of the protocol

Figure 1 sketches the basic protocol for proving an attribute that was used in [1]. To start, the terminal requests the card to show its credentials (1). The card responds (3) with its attribute, certificate and public key. The terminal can verify that the card actually possesses this attribute by verifying the certificate (4) which links the attribute to the public key. Finally, in steps (5) to (8), the terminal verifies, using a standard challenge-response mechanism, that the card actually knows the private key corresponding to the public key. These two parts can be combined to reduce the amount of messages sent.

What makes this protocol privacy friendly is the blinding of the card specific, and thus identifying, values, performed in step (2). By using a fresh blinding for each run the terminal will be unable to link a protocol run to a previous, or future, one.

3.2 The Optimised Protocol

We will now present our optimised version of the above protocol.

System Setup The scheme provider has a public fixed point Q and a finite set of attributes. For each attribute a a secret key s_a and public key $Q_a = s_a \cdot Q$ are generated. The associated pairs (a, Q_a) of attributes and public keys are publicly known, and stored in all terminals together with the fixed point Q.

A card c generates a key pair $k_c, P_c = k_c \cdot P$ where P is a fixed system wide generator. The private key k_c of the card is assumed to be stored in a protected manner such that it cannot leave the card. Upon personalisation the card receives its attribute a together with a corresponding certificate $C_a = s_a \cdot P_c$ linking its public key P_c to the attribute a. The attribute a corresponds to a product that the owner of the card has bought.

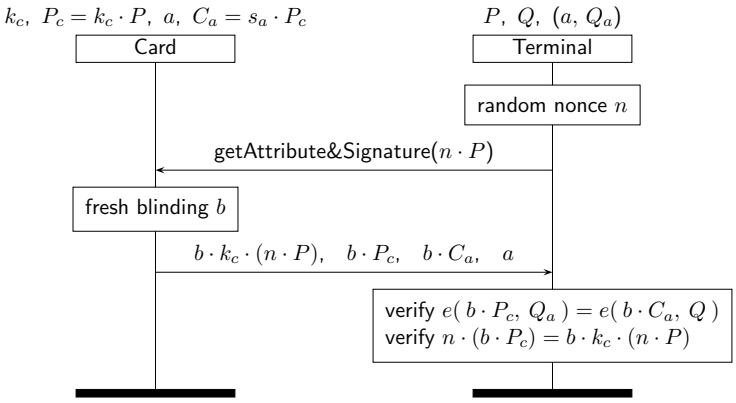

Figure 2. Optimised protocol for proving self-blindable attributes

Protocol Description The protocol for proving self-blindable attributes, as depicted in Figure 2, is initiated by a terminal which sends a request, together with a point representation of the nonce $n \cdot P$, to the card. The card generates a fresh blinding factor b to blind its key pair and the certificate. It responds by sending a signature $b \cdot k_c \cdot (n \cdot P)$ of the received nonce, which is created using its blinded private key $b \cdot k_c$, and the blinded values ($b \cdot P_c$ and $b \cdot C_a$) together with the attribute a stored on the card.

The terminal can now perform a pairing signature verification, as discussed in Section 2, using the card's response, the attribute's public key Q_a and the fixed point Q. Note that the terminal can select the correct public key Q_a to use by matching the attribute returned by the card. Finally the terminal verifies the signature using the blinded public key and the scalar nonce value n. When the verification succeeds the card has proved possession of the requested attribute.

3.3 The Implementation

Java Card Applet In [1] the practical limitations of Java Cards have been described that have to be taken into account while programming the card. The actual operations that the card needs to perform are scalar multiplication of points. In the end the applet performs the required steps of the protocol in the following way.

The difficulty in the Java Card applet is the blinding of the private key. This problem has been circumvented by (ab)using the EC key generation operation to generate the blinding factor. This function generates a random number which it multiplies with the generator point of the elliptic curve. By setting the nonce, received from the terminal, as the generator this function produces a private key (the blinding factor b) and a public key (the blinded nonce $(b \cdot (n \cdot P))$ which we can use for the remaining calculations.

Two EC Diffie-Hellman (ECDH) key agreement operations (effectively two scalar point multiplications) are performed with this generated private key to calculate the blinded public key, and the blinded certificate. A third ECDH key agreement is performed using the generated public key, the blinded nonce, together with the cards private key to generate the signature.

Both operations, key generation and key agreement, use the cryptographic coprocessor to perform the necessary calculations. This improves performance since there are no longer calculations which have to be done in software, which was the drawback from the original implementation.

Terminal Application The implementation of the terminal application is not significantly different from the original version. It only contains minor modifications to incorporate the integration of the challenge-response part in the first messages and the new signature check.

3.4 The Results

Using the same parameters as Batina et al. [1] we have tested the performance of our optimised version of the protocol. The results of these tests can be found in Table 1. The results of the original protocol have been included, for comparison, in Table 2. Note that we no longer use different blinding lengths since we now use the key generator for this, which just uses the elliptic curve parameters.

Table 1. Test results of the optimised protocol for various key lengths

key length (bits)	attribute & signature (ms)	verification (ms)	protocol total (ms)	communication (bytes)
192	787	116	904	155
160	645	102	747	135
128	535	82	617	115

Table 2. Test results of the original protocol for various key and blinding lengths

key (bits)	blinding (bits)	attribute & signature (ms)	verification (ms)	protocol (ms)	communication (bytes)
192	192	2748	143	2891	168
	96	1884	136	2020	
	48	1451	130	1582	
160	160	1860	126	1987	152
	80	1355	133	1489	
	40	1113	127	1240	
128	128	1599	91	1691	136
	64	1143	93	1237	
	32	927	86	1014	

It can be seen that there is a significant increase in performance with respect to the running times on the card from the original implementation. The smaller amount of data which has to be exchanged during communication allowed us to combine the protocol in a single command-response APDU pair, thus reducing the amount of messages sent which also has a positive effect on the processing overhead.

To get some more information about how the running time is spent on the card we measured how long it takes to perform the individual operations. The results of these measurements can be found in Table 3. The columns indicate the time needed to perform a single operation. The processing overhead is determined by subtracting one key generation and three key agreements from the running time on the card.

Table 3. Test results for the API primitives

key length (bits)	key generation (ms)	key agreement (ms)	processing overhead (ms)
192	379	98	114
160	307	78	104
128	242	62	107

On the one hand, performing a scalar point multiplication (a key agreement) is quite efficient, using less then 100 ms for the calculation. On the other hand, performing a scalar point multiplication, combined with generating a random value, (a key generation) is disappointing, taking more than a factor three longer than just the multiplication. A possible explanation is a different calcula-

tion which explains the fact that a key generation can return a complete point, whereas a key agreement can only return the x-coordinate.

4 Conclusions

This short note presents another small step on the way to making anonymous credentials usable in the context of public transport with its tight performance challenges. Further optimisations are still needed to meet the requirements. These improvements could result from (a combination of):

- faster smart card hardware;
- access to the crypto-coprocessor through a full-fledged crypto-API that gives access to exactly the functions one needs;
- lower level implementations, not using Java as implementation language, but some machine language for more direct access to the processor and cryptographic coprocessor on the card.

The third step is the most obvious one, but means that one has to give up the high-level card-independent feature of Java Card. Another hindrance is that card producers do not easily give direct access to the card hardware, or only under very severe non-disclosure agreements (NDAs) that make it difficult to publish any results.

References

1. Batina, L., Hoepman, J.H., Jacobs, B., Mostowski, W., Vullers, P.: Developing efficient blinded attribute certificates on smart cards via pairings. In: Gollmann, D., Lanet, J.L. (eds.) Smart Card Research and Advanced Applications, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 13-16, 2010. Proceedings. LNCS, vol. 6035, pp. 209–222. Springer (2010)
2. Garcia, F.D., de Koning Gans, G., Muijrers, R., van Rossum, P., Verdult, R., Wichers Schreur, R., Jacobs, B.: Dismantling MIFARE Classic. In: Jajodia, S., Lopez, J. (eds.) 13th European Symposium on Research in Computer Security (ESORICS 2008). LNCS, vol. 5283, pp. 97–114. Springer (2008)
3. Garcia, F.D., van Rossum, P., Verdult, R., Wichers Schreur, R.: Wirelessly pickpocketing a Mifare Classic card. In: IEEE Symposium on Security and Privacy (S&P '09). pp. 3–15. IEEE (2009)
4. Verheul, E.: Self-blindable credential certificates from the Weil pairing. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001. LNCS, vol. 2248, pp. 533–550. Springer (2001)