SOK it to the IoT

Michael Scott and Kealan McCusker

MIRACL Labs
mike.scott@miracl.com
September 14, 2016

Abstract. The non-interactive authenticated key exchange protocol known as SOK after its inventors Sakai, Oghishi and Kasahara, is one of the original pairing-based protocols. Here we suggest that it makes an ideal bootstrapping mechanism for the Internet of Things. As it was originally proposed, it was designed to work with a symmetric pairing. However now it is known that symmetric pairings are very inefficient. So the issue arises of how to migrate it successfully to the setting of an efficient asymmetric pairing. Here we consider the challenges and opportunities. Since pairings are often regarded as being too resource demanding for small processors, we also make available a small foot-print portable library which is tuned for pairing-based cryptography. Finally we propose a distributed trusted authority infrastructure to eliminate any single point of failure from the system.

1 Introduction

The SOK protocol [11] proposes the only known practical method for non-interactive authenticated key exchange. As originally described it is based on a type-1 pairing [6] on a supersingular elliptic curve. A type-1 pairing operates as $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ is a group of points of prime order q on the curve, and $\mathbb{G}_T$ is a finite extension field of the same order, whose extension is the so-called embedding degree k associated with the curve. The SOK inventors were one of the first to realise that by carefully matching the field size with the embedding degree, that the discrete logarithm problem could remain hard in both $\mathbb{G}_1$ and $\mathbb{G}_T$, and hence that the pairing was a suitable vehicle for cryptography. Recently NIST endorsed pairing-based cryptography [9].

A type-1 pairing has the property of symmetry, and it turns out that this property is quite important to the SOK protocol as originally described. However time has not been kind to type-1 pairings over the intervening years. For required levels of security either $\mathbb{G}_1$ or $\mathbb{G}_T$ must be greater than strictly necessary due to the restricted choice of embedding degree possible on supersingular curves, leading to inefficiencies. And for some of the most promising families of supersingular curves, it turns out that the discrete logarithm problem in $\mathbb{G}_T$ is much easier than originally expected [7].

The most efficient pairing is the asymmetric type-3 pairing, which works with non-supersingular pairing-friendly curves. These operate as $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$,

where $\mathbb{G}_2$ is a particular group of points, again of the order q, but on a twisted elliptic curve defined over an extension which is a divisor of k. These curves can be constructed to be a good fit at any required level of security [5]. Until recently the BN curves [2], with an embedding degree of $k = 12$ were regarded as an exact fit for the AES-128 equivalent level of security. Progress in the analysis of the discrete logarithm problem that arises in the finite field $\mathbb{G}_T$ [8], now suggest that somewhat larger curves must be used.

Pairings are usually written as functions of the form $g = e(A, B)$, where $A \in \mathbb{G}_1$, $g \in \mathbb{G}_T$, and for a type-1 pairing $B \in \mathbb{G}_1$ and for type-3 $B \in \mathbb{G}_2$. In both cases there are efficient ways to hash an arbitrary string to an element in $\mathbb{G}_1$ or $\mathbb{G}_2$, but we omit the details.

We should also mention the type-2 pairing, which allows a pairing between a pair of elements in $\mathbb{G}_2$. But to preserve the symmetry property for the SOK protocol it would have to be possible to hash arbitrary strings into the same group of order q in $\mathbb{G}_2$, and there is no known way to do this [6].

2 The SOK protocol

The SOK protocol showcases the bilinearity property of the pairing

$$e(xA, B) = e(A, xB) = e(A, B)^x$$

For a type-1 pairing there is also the property of symmetry

$$e(A, B) = e(B, A)$$

We first assume a type-1 pairing is being used. A Trusted Authority generates a master secret s. Alice and Bob separately visit the trusted authority, and present their identities and prove their right to those identities in some way. Alice is issued with her secret $s.A$ where $A = H_1(\text{“Alice”})$, and the hash function $H_1(.)$ hashes the identity string to a point of order q in $\mathbb{G}_1$. The trusted authority calculation is simply the well known operation of multiplication by the scalar s of a point on an elliptic curve. Similarily Bob is issued with the secret $s.B$ where $B = H_1(\text{“Bob”})$.

Now Alice and Bob can communicate using a shared authenticated key, calculated by Alice as $e(sA, B)$ and by Bob as $e(sB, A)$. These keys will be the same due to bilinearity and symmetry. Note that by convention each can put their own secret as the left-hand argument of the pairing, and the hashed identity of the other as the right-hand argument.

2.1 Relevance to the IoT

Consider now an application of this protocol to an imagined Internet of Things (IoT) setting as first proposed in [10]. Each Thing is issued with a serial number and its own SOK secret based on that serial number as an identity. These SOK

secrets may be embedded at the time of manufacture, by the manufacturer acting as a naturally trusted authority.

When a Thing needs to communicate with another Thing, an action which requires knowing only the identity of the other, both parties can activate SOK to calculate the same key to encrypt and authenticate their communication.

However in reality this description is probably a massive simplification of a real world IoT deployment. Are the Things capable of protecting their secrets from an attacker, do they support secure storage? Are they capable of calculating a pairing? Do they communicate on a peer-to-peer basis, or client-server? Are the things mobile or stationary, what is the network topology of their communication links, and are they fixed or fluid? Are all Things created equal or do some have more resources than others? Since we do not have a particular application in mind, we merely suggest that SOK might be a nice fit for at least some of these scenarios.

3 Migrating to a type-3 pairing

Take away the symmetry property, and things get a bit more complicated. One thing we can exploit – in any communication context there is an initiator and a responder. Therefore the obvious solution is to issue each entity with two secrets, one in $\mathbb{G}_1$ and the other in $\mathbb{G}_2$, as proposed by Dupont and Enge [4]. So Alice is issued with sA_1 and sA_2, where $A_1 = H_1(\text{“Alice”})$ and $A_2 = H_2(\text{“Alice”})$. We call these Alice's lefthand and righthand secrets respectively, as this describes where they can appear in the pairing. Similarly Bob is issued with sB_1 and sB_2. Now if Alice initiates and Bob responds, Alice calculates the key as $e(sA_1, B_2)$ and Bob can calculate the same key as $e(A_1, sB_2)$, where by convention the initiator uses their lefthand secret and the responder uses their righthand secret.

4 An IoT deployment

That seems an appropriate and workable solution. However maybe we can do better. Consider again the IoT setting. Now Things are divided into two categories, Talkers and Listeners. Some Things might have only one of these attributes, some may have both. But now this division of capabilities can be cryptographically enforced, by issuing lefthand secrets only to talkers and righthand secrets only to listeners. Perhaps a listener-only Thing might be lower powered, and perhaps its secret does not need to be so vigorously defended, as a hacked listener secret may be of less significance.

At first glance it may appear that a listener secret could still be used to talk by exploiting bilinearity – if we cannot calculate $e(sA_1, B_2)$ because we do not possess sA_1, we could instead calculate $e(B_1, sA_2)$. But these are not the same as $A_1 \neq A_2$ and $B_1 \neq B_2$.

Without making any dogmatic claims, we suggest that this attribute of SOK on a type-3 pairing may in fact be considered as a useful feature in many IoT contexts.

Given the undoubted resource consumption involved in calculating a pairing, it is probable that the pairing will only be invoked on first-contact between Things, after which the agreed key might be cached for future use, or used to bootstrap up into something more efficient. However an IoT deployment is probably at its most vulnerable during its initialisation, and for this our proposal offers cryptographic security in the form of an encrypted and authenticated channel between Things, available from the very start. We also recognise that this proposal is for use only at the lowest layers of a security stack. We would visualize that more elaborate higher-level protocols would be built on top of it.

5 A software library

Much has been written on the computational effort involved in pairing-based cryptography. Unfortunately some early and still widely-used libraries are very inefficient and slow, and do not make use of the very latest optimizations. So to realise our IoT offering we first set about developing an efficient IoT-friendly software libary. The Apache Milagro Crypto Library (AMCL) [1] is portable and truly multi-lingual as it only uses generic programming constructs. It is currently is available in C, Java, C#, Javascript, Go, Rust and Swift. Here we focus on the C version. AMCL is small and takes up the minimum of ROM/RAM resources in order to fit into the smallest possible embedded footprint, consistent with other design constraints. AMCL only uses stack memory, and is thus natively multi-threaded.

AMCL supports AES/128/192/256 for symmetric encryption, and SHA256/384/512 for hashing. Standard modes of AES are supported, plus GCM mode for authenticated encryption. Prime field elliptic curves are supported for public key protocols, and BN [2] and BLS [1] curves to support pairing-based protocols, like SOK. Three different parameterizations of Elliptic curve are supported - Weierstrass, Edwards and Montgomery, as each is appropriate within its own niche. In each case the standard projective coordinates are used. The user can choose the actual elliptic curve, with support for three different forms of the modulus.

AMCL is configured at compile time for 16, 32 or 64 bit processors, and for a specific elliptic curve. The library is written with an awareness of the abilities of modern pipelined processors. In particular there was an awareness that the unpredictable program branch should be avoided, not only as it slows down the processor, but as it may open the door to side-channel attacks. The innocuous looking `if` statement – unless its outcome can be accurately predicted at runtime – is the enemy of quality crypto software.

No external libraries or packages are required to implement all of the supported cryptographic functionality (other than for an external entropy source). There is included a basic X.509 module as we recognize the need to support X509 standards (not out of conviction, as we are strongly of the view that PKI is not appropriate for the IoT, but for legacy reasons).

[1] `https://github.com/MIRACL/amcl.git`

5.1 Representing Big Numbers

As is well known to support elliptic curve and pairing-based cryptography efficiently, it is important to be able to do basic modular arithmetic on numbers whose size exceeds the word-length w of the processor. Therefore such numbers must be represented using a fixed number of words. Here we went with a "reduced radix" rather than a "packed-radix" representation, where each word represents a digit of the larger number considered as being to the base 2^b, where b is a few bits less than w. This method is often used, see for example [3]. It facilitates efficient portable implementation (carry bits do not need to be handled), encourages efficient compilation, allows faster multiplication [12], supports lazy reduction, and assists in the generation of side-channel resistant code.

5.2 Performance

A pairing on a BN curve at the AES-128 level of security on a Raspberry Pi computer (32-bit, Version 1, 700MHz), which is often touted as an IoT platform, can be calculated using this software in just 86ms. We regard this as quite acceptable for bootstrapping purposes. To implement the SOK protocol the code required 85k of ROM and 8K of RAM (stack memory). To respond to recent developments [8], we also implemented SOK on a 455 bit BLS [1] curve on the same platform. The pairing was calculated in 205ms, and the stack requirement rose to 16k.

6 A Distributed Trust Authority Infrastructure

In our view one of the best way to manage trust, is to distribute it. Secure Multi-Party Computation is often touted as being the best way to distribute cryptographic trust. However performing MPC on cryptographic primitives that have not been designed with MPC in mind, can be very challenging. Fortunately it turns out that the form of secrets often used in pairing-based cryptography, and used here, are ideal for MPC.

Here the problem we are addressing is the Trusted Authority as a potential single-point-of-failure. Our solution to this problem is to distribute the TA functionality, and establish a D-TA infrastructure. For example a pair of D-TAs can generate their own independent secrets s_1 and s_2, and separately issue s_1A and s_2A to Alice. Alice simply adds these components together to form her full secret.

To this end we have built a scalable cloud-based DTA infrastructure. Any number of DTAs can be used in a particular application. When a new customer (for example an IoT manfacturer) registers with MIRACL, a D-TA is started on their behalf on the MIRACL infrastructure which uses the Amazon Web Service (AWS) platform. The partial master secret is then encrypted with an AES key that is secured using the AWS key management service, which is built using hardware security modules. This approach enables MIRACL to horizontally scale the D-TAs with ease. The customers are also provided with the code to run their own independent D-TA.

References

1. P.S.L.M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks – SCN 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer-Verlag, 2003.
2. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC'2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer-Verlag, 2006.
3. Daniel J. Bernstein, Chitchanok Chuengsatiansup, and Tanja Lange. Curve41417: Karatsuba revisited. Cryptology ePrint Archive, Report 2014/526, 2014. `http://eprint.iacr.org/2014/526`.
4. R. Dupont and A. Enge. Practical non-interactive key distribution based on pairings. Cryptology ePrint Archive, Report 2002/136, 2002. `http://eprint.iacr.org/2002/136`.
5. D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptography*, 23:224 – 280, 2010.
6. S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.
7. R. Granger, T. Kleinjung, and J. Zumbragel. Breaking 128-bit secure supersingular binary curves. In *Advances in Cryptology – Crypto 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 126–145. Springer-Verlag, 2014.
8. T. Kim and R. Barbelescu. Extended tower number field sieve: A new complexity for the medium prime case. Cryptology ePrint Archive, Report 2015/1027, 2015. `http://eprint.iacr.org/2015/1027`.
9. D. Moody, R. Peralta, R. Perlner, A. Regenscheid, A. Roginsky, and L. Chen. Report on pairing-based cryptography, 2015. `http://nvlpubs.nist.gov/nistpubs/jres/120/jres.120.002.pdf`.
10. L.B. Oliveira, D.F. Aranha, C.P.L. Gouvea, M. Scott, D.F. Camara, J. Lopez, and R. Dahab. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34:485–493, 2011.
11. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.
12. M. Scott. Missing a trick: Karatsuba variations. Artic Crypt, 2015. `http://eprint.iacr.org/2015/1247`.