Distributing SAKKE Private Key Generation

Michael Scott

Certivox Labs
Invent Centre
Dublin City University
Ballymun, Dublin 9, Ireland.
mike.scott@certivox.com

Abstract. The Sakai-Kasahara IBE scheme [9] (also known as SAKKE) uses an "inverse exponent" method to generate private keys from identity strings. This process is carried out by a Key Distribution Centre (KDC). The point P_2 on the pairing-friendly elliptic curve is a fixed system parameter, and the private key issued to a client with identity ID is $K = 1/(x + ID).P_2$, where x is the KDC's master secret. However all IBE schemes are vulnerable to "key escrow", whereby the KDC, knowing x can itself decrypt all client ciphertexts. The recommended solution is to distribute the KDC functionality across multiple Servers. In essence each Server holds only a portion of x, useless on its own. But this is a non-trival task. A clever solution based on ideas from secure multi-party computation has been proposed by Geisler and Smart [7]. Here we walk through the simplest possible realisation of the Geisler-Smart algorithm for distributing the Key Distribution Centre in Sakai-Kasahara based IBE systems. We propose an architecture which minimizes the complexity, and fully implement it.

1 Introduction

In the simplest case an IBE master secret x could be split two ways, with half given to each of a pair of KDCs. This could for example be the case for the Boneh-Franklin IBE scheme [1]. However because of the nature of the inverse exponent calculation required for Sakai-Kasahara, here we need a minimum of three.

In the notation of Geisler-Smart [7] we are interested in the case $t = 1$, $n = m = 3$. We will use three Servers denoted Master Server 1, Server 2 and Server 3 respectively. The master secret x is to be distributed between these three Servers. Any single Server on its own cannot decrypt client ciphertexts, although a coalition of two or more can.

The Geisler-Smart paper [7] is not very explicit on implementation, and some of the references are rather opaque and highly mathematical. Here we describe in detail a simple architecture for a practical implementation.

First we describe the Sakai-Kasahara method [9] in a little more detail.

Although this protocol can be implemented on either a type-1 or type-3 pairing [6] we will assume here the use of a type-3 pairing $G_1 \times G_2 \rightarrow G_T$

where for example $P_T = e(P_1, P_2)$ and $P_1 \in G_1$, $P_2 \in G_2$, and $P_T \in G_T$. P_1 and P_2 are points on special elliptic curves, and P_T is an element of a finite extension field. All members of each of these three groups are of prime order q. The most important property of the pairing is its bilinearity, where $e(aP_1, bP_2) = e(bP_1, aP_2)$. Exploiting this, Sakai-Kasahara go on to develop an IBE scheme which does not require the calculation of any pairing for encryption, and just one pairing for decryption. We omit the full details, as they are not relevant here. In a non-distributed Sakai-Kasahara IBE the client's private key $K = 1/(x + ID).P_2$ is issued by a Key Distribution centre, where P_2 is a fixed public point and ID is the identity of the client.

There is also a requirement in this protocol to provide to each client the public key $R = xP_1$.

2 Secret Sharing

Shamir's secret sharing scheme [10] is a little bit like magic. Take the secret x and generate the classic equation of a straight line $f(X) = mX + x$, where m (the slope) is chosen at random. Note that x is the intercept of the line with the vertical axis. Then given the two "shares" in x, say $x_2 = f(2)$, $x_3 = f(3)$, x can be recovered as $3x_2 - 2x_1 = 3(2m + x) - 2(3m + x) = x$. This is independent of the choice of m, and is called Lagrangian interpolation. Note that x cannot be recovered given only one share.

3 Distributing the Public Key

Having chosen a random line equation m, x can be divided into three Shamir shares x_1, x_2 and x_3, from which x can be reconstructed from any two of these using Lagrangian interpolation. One share is given to each of the three servers.

Servers 2 and 3 for example can each individually calculate a Shamir share of the public key $R_i = x_i P_1$ and pass this to the client. The client can combine these shares to obtain the public key $R = 3R_2 - 2R_3$. It is not so easy to distribute calculation of the more complex inverse exponent form of an individuals private key.

4 Distributing the Master Secret

The above simple description assumes the existence of a "dealer", someone who distributes the shares of x as x_i to each of the three Servers. But such a dealer obviously knows x, and this rather defeats the purpose, as now the dealer has escrow capabilities. Somehow we need to get the three Servers to each independently calculate their own share of an unknown shared secret without the help of a dealer.

So the three Servers need to each independently come up with 3 random points on a random line. However 3 independently chosen points will not be co-linear. So clearly the solution is going to require some communication between the Servers.

The proposed solution is for each pair of Servers to carry out a once-off initial Diffie-Hellman key exchange [5]. If the Servers are numbered 1, 2 and 3, then the pairs are the sets $A = \{2, 3\}$, $B = \{3, 1\}$ and $C = \{1, 2\}$. As a result of running Diffie-Hellman each set has possession of three values x_A, x_B and x_C. The idea is that the shared secret is $x = x_A + x_B + x_C$. Clearly each individual Server is not in a position to recover x, as each is missing one value from this sum.

However this is not Shamir secret sharing as these are not 3 points on a line. Consider the line $f(X) = m.X + x$ which $m = -x_A - x_B/2 - x_C/3$. Now Server 1 can calculate its point as $x_1 = f(1) = x_B/2 + 2x_C/3$, Server 2 can calculate its point as $x_2 = f(2) = -x_A + x_C/3$, and Server 3 can calculate its point as $x_3 = f(3) = -2x_A - x_B/2$. Note that each Server has the information needed to make these individual calculations. It is easy to verify that the value of x can be recovered by any pair of Servers, as for example $x = 2x_1 - x_2$.

The idea can readily be extended to set up more than 3 Servers. This may be regarded as building in redundancy as an insurance against one or more Servers crashing. However only 3 active Servers are actually needed at any given time.

Basicly to set up m Servers, each subset of $m - 1$ Servers carries out a group key exchange (using for example the methods of [2] or [3]). The master secret is the sum of all these secrets. The sets are created as $A = \{2, 3, 4...\}$, $B = \{3, 4, 5..\}$ etc. For example after the members of set A have completed their group exchange they all possess x_A. The master secret is $x = x_A + x_B + x_C...$, and the line equation is $f(X) = mX + x$ where $m = -x_A - x_B/2 - x_C/3....$ Note that each Server is still short one share in the master secret. Finally server n calculates its point on the line as $x_n = f(n)$.

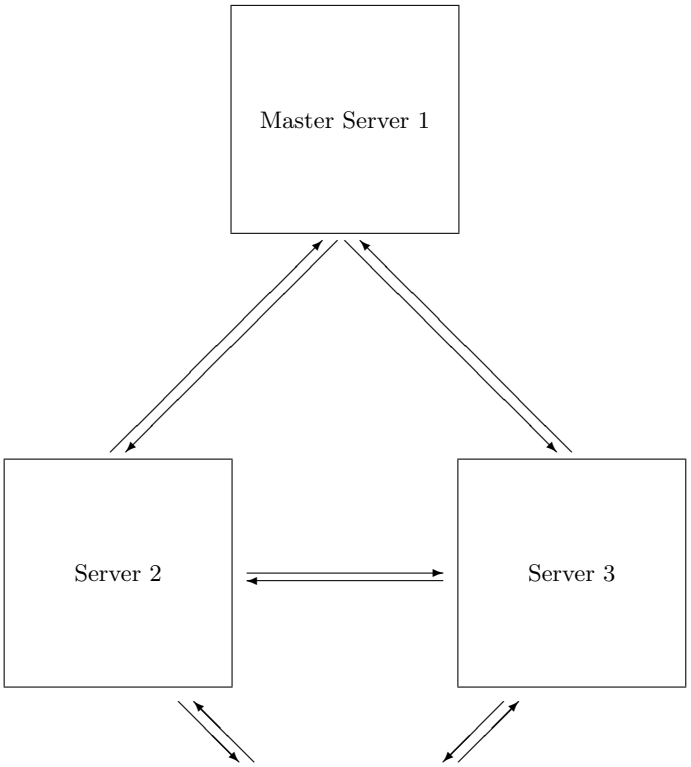
5 Proposed architecture

It is annoying that three Servers are needed, so we will do our best to minimise the inconvenience of this. See Figure 1 below. Essentially our plan is that when the client is to be issued with a private key associated with their identity, they only need to approach two of the servers and not all three. The "extra" third server sits in the background, and only communicates with the two "front-of-house" servers who deal with clients. The so-called Master Server might be implemented in secure hardware, it could have limited exposure to the Internet, and in that way it might be made an unattractive focus for attack.

6 Distributing private keys

The client with identity ID approaches Servers 2 and 3 in turn. The identity ID is passed back to Server 1. The i-th server calculates their share of $z = x + ID$ as $z_i = x_i + ID$.

Fig. 1. Proposed Architecture



To proceed securely, a secret random value r must now be shared between all three nodes. Such a number could be generated just as for the distribution of the master secret x. But since we have already done that work, there is an easier way to proceed. The Master Server generates a random 256-bit value W and broadcasts it to Servers 2 and 3. The Servers in each set now calculate $r_A = H(x_A|W)$, $r_B = H(x_B|W)$ and $r_C = H(x_C|W)$, and from these values each individually calculates their Shamir share in r as r_1, r_2 and r_3, just as above. The function $H(.)$ can be based on a hash function like SHA256, where the symbol | means concatenation. This construction is known in the literature as a PRSS – a Pseudo Random Secret Sharing.

Next is the tricky bit [4] – each of the Servers needs to be able to calculate $s = z.r$ in a secure distributed manner between the three servers. To start each calculates $s_i = z_i.r_i$.

Master Server 1 generates a random line equation and calculates two Shamir shares in s_1, namely s_{12} and s_{13}. These are sent to Servers 2 and 3 respectively. Server 2 then generates a random line equation and calculates two Shamir shares for s_2 as s_{22} and s_{23}, and sends the latter to Server 3. Finally Server 3 generates a random line equation and calculates two Shamir shares in s_3 as s_{33} and s_{32}, and sends the latter to Server 2.

Server 2 then calculates its Shamir secret share of s as $ss_2 = 3s_{12} - 3s_{22} + s_{32}$. Server 3 calculates its Shamir secret share of s as $ss_3 = 3s_{13} - 3s_{23} + s_{33}$. Servers 2 and 3 swap these values. Each can now calculate the full $s = 3ss_2 - 2ss_3$.

Finally Servers 2 and 3 each calculate their share of the final value of the private key $1/(x+ID)$ as $k_2 = r_2/s$ and $k_3 = r_3/s$ respectively. When a client ID comes calling, they visit both Servers 2 and 3 and are issued with $K_i = k_i P_2$. These values can be combined by the client to reveal the full private key as $K = 3K_2 - 2K_3 = 1/(x + ID)P_2$

The client can check the validity of their key by exploiting bilinearity and checking that $e(R + ID.P_1, K) = e(P_1, P_2)$.

7 Implementation

In [7] there is a description of an implementation using a special purpose framework for multi-party computation. Here we describe a more practical implementation in C++ using TCP/IP sockets to connect the Servers and the client. The three Servers and a client are run as separate processes on a 4-core Intel i5 520M clocked at 2.4GHz. The implementation uses our MIRACL library and the version of SAKKE described in a recent IETF RFC [8].

The computation involved is not very onerous, mostly simple group operations. The only exception might be on the client side, as the client is required in [8] to check the validity of the private key issued to it. This requires a very expensive pairing calculation.

Our first test was to time how long it took to issue 1000 private keys. The client process approached Servers 2 and 3 to extract and test private keys for 1000 different identities. This took 28 seconds. By modifying the client so that

it did not test its private key each time, this fell to 10 seconds. Note that key testing would normally be done off-line anyway, and so this is a fairer measure of performance. It appeared that each core was only operating at 50% of full load, which probably indicates that each server is spending up to half of its time waiting for input.

Note that unlike [7] we give timings for the whole process, including the relatively expensive point multiplications. Also note that for the SAKKE scheme as described in [8] the field is of size 1024 bits, which makes point multiplication particularly expensive.

We concludes that it takes about 10ms for each client to be issued with a private key. This is entirely practical.

8 Discussion

It may be asked, surely it is possible to use two Servers instead of three? No its not, as three is a requirement of the distributed method of securely calculating the product $s.r$. Observe how the random r_i values mask transmitted data.

The inter-Server communications needed for this protocol do not need to be encrypted in any way. The original Diffie-Hellman calculation between pairs of nodes might appear to be a weakness, as a Man-In-The-Middle (MITM) attack can potentially be deployed again it. However this is just done once ever during the setting up of KDCs, and it is not considered a practical risk.

As indicated above, shares in the Public key R can be provided directly by Servers 2 and 3 without involving Master Server 1.

The need for a third Server may be seen as a disadvantage, but given that we have to have it, we have done our best to keep its involvement to a minimum. We emphasise that the master secret x is not known at any time, even fleetingly, to any individual party in this protocol, and no single entity can calculate it. So if a single Server is compromised, then this will not allow the recovery of the master secret x. In this way we effectively side-step the key-escrow issue.

References

1. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
2. E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater. Provably authenticated group diffie-hellman key exchange. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 255–264, 2001.
3. M. Burmester and Y. Desmedt. A secure and scalable group key exchange system. *Information Processing Letters*, 94(3):137–143, 2005.
4. R. Cramer, I. Damgard, and U. Maurer. General secure multiparty computation from any linear secret sharing scheme. In *Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer-Verlag, 2000.
5. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.
6. S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008.
7. M. Geisler and N. Smart. Distributing the key distribution centre in sakai-kasahara based systems. In *Cryptography and Coding*, volume 5921 of *Lecture Notes in Computer Science*, pages 252–262. Springer-Verlag, 2009.
8. M. Groves. Sakai-kasahara key encryption (SAKKE). IETF RFC 6508, 2012.
9. R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054, 2003.
10. A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

9 Appendix

Here we walk through an example.

Assume the active servers are $S = 1, 2, 3$, and that each one's share of a value x is defined as a point on the line $y = Sm + x$, for some randomly chosen slope m.

For each server to calculate a share of the point $1/(x + ID)P_2$, proceed as follows. Each has a share z_i of $x + ID$ and and a share r_i of a random number r. So

$$r_1 = m + r, \quad z_1 = n + (x + ID)$$
$$r_2 = 2m + r, \quad z_2 = 2n + (x + ID)$$
$$r_3 = 3m + r, \quad z_3 = 3n + (x + ID)$$

These shares are multiplied, and shares of their product are calculated by server 1 as.

$$c_{11} = w + (m + r)(n + x + ID)$$
$$c_{12} = 2w + (m + r)(n + x + ID)$$
$$c_{13} = 3w + (m + r)(n + x + ID)$$

By server 2 as

$$c_{21} = u + (2m + r)(2n + x + ID)$$
$$c_{22} = 2u + (2m + r)(2n + x + ID)$$
$$c_{23} = 3u + (2m + r)(2n + x + ID)$$

By server 3 as

$$c_{31} = v + (3m + r)(3n + x + ID)$$
$$c_{32} = 2v + (3m + r)(3n + x + ID)$$
$$c_{33} = 3v + (3m + r)(3n + x + ID)$$

These are distributed to the other servers.

Server 1 now uses the interpolation formula on c_{11}, c_{21} and c_{31} to recover a share of the product $r(x + ID)$.

$$p_1 = 3c_{11} - 3c_{21} + c_{31} =$$
$$3(w+(m+r)(n+x+ID))-3(u+(2m+r)(2n+x+ID))+v+(3m+r)(3n+x+ID)$$

And by simple manipulation this simplifies to

$$p_1 = 3w - 3u + v + r(x + ID)$$

Observe how all other terms cancel out. To achieve this cancellation it is clear that a minimum of 3 shares are required. Similarly the other servers calculate

$$p_2 = 2(3w - 3u + v) + r(x + ID)$$
$$p_3 = 3(3w - 3u + v) + r(x + ID)$$

Clearly these represent 3 shares of the product $r(x + ID)$ on a line of slope $3w - 3u + v$. These shares are distributed to the other servers who combine them to recover $r(x + ID)$.

$$r(x + ID) = 3p_1 - 3p_2 + p_3$$

Each then divides their share of r by this value, and uses it to multiply P_2 to create a share of the key K.

$$K_1 = (m + r)/r(x + ID)P_2$$
$$K_2 = (2m + r)/r(x + ID)P_2$$
$$K_3 = (3m + r)/r(x + ID)P_2$$

The client can combine these shares to find K as

$$K = 3K_1 - 3K_2 + K_3 = r/r(x + ID)P_2 = 1(x + ID)P_2$$

As pointed out above this recombination actually only requires 2 shares, as for example

$$K = 3K_2 - 2K_3$$