622.755 – Crypto Engineering Winter Term, 2020/21

Project: SuperSAM Implementation Note — Internal, Confidential, For YOUR Eyes Only

Elisabeth Oswald

Please Read Carefully

This document describes a "mock up" implementation note supposedly relating to the design of a secure key fob to be used within a large organisation. **Under no circumstances should this document ever be used as a template for a "real" implementation note, or be understood as guidance towards designing a secure key fob.** On the contrary, this document, whilst in style and formalism being similar to notes that I have reviewed in real life, has been deliberately laced with bad practice, serious errors, and minor (but hugely consequential) mistakes. It serves as an example ONLY in the context of the "Cryptographic Engineering" course, with the purpose of enabling students to spot bad practice, find errors and mistakes, and eventually suggest fixes.
Similarly, the related implementation has been deliberately designed to include significant mistakes. Please do not base anything on it.

1 Introducing SuperSAM

This note describes the considerations and design of our new Super Secure Access Module (SuperSAM) as part of the Secure Access Strategy (SAS) put forward by the chief security officer. The intention of the SAS is to deploy a fully integrated access system for employees with in the organisation; this should in particular include physical access to buildings/infrastructure.

The vision of the SAS is that only employees can gain access to physical buildings. SuperSAM is the technical enabler for this vision: upon starting with the company (or for existing employees upon rollout of SuperSAM) every employee will receive their own SuperSAM access token (in form of a small key fob). Via a proximity based RFID system, presenting the SuperSAM token will enable employees to access certain buildings. The big advantage of this system is that any SuperSAM token will store information about their respective access rights (in form of a ticket); the reader only verifies but does not have to consult a central database. Therefore our system **guarantees that employees can always access the buildings that they should have access to**.

2 System Overview

Whilst there has been no formal review of all the requirements so far, we assume that an implementation of an asymmetric and an implementation of a symmetric cryptographic primitive will be required.

We also assume that each user will have only one SuperSAM token: the token needs to securely store the information about the buildings that the user has access to.

SuperSAM tokens need to communicate with reader devices at close proximity. Therefore we require lightweight cryptographic primitives and lightweight random number generation.

2.1 Token capabilities

A SuperSAM token has all the information that it needs to enable employee access to specific buildings (this information is stored in a data structure that we name *ticket*). A SuperSAM token has therefore information that to prove that it is authentic, and it has functionality to establish a secure channel with the reader.

To prove authenticity we assume that each token holds some ID encrypted under the system master key (SMK). The token has cryptographic functionality agree on a session key with the reader via a Diffie-Hellman key exchange over a prime field with large order.

We will use the strongest known encryption algorithm for storing data on the token: the Advanced Encryption Standard (AES).

2.2 Reader capabilities

A reader needs to verify the authenticity of a SuperSAM token (via checking the supplied encrypted ID). It needs to establish a secure channel and agree with a token on a key and decrypt ticket information.

There is a special class of readers in the field called Master Readers. They have the ability to update tickets on tokens. These reader will have, in addition to the normal functionality, an update function implemented.

2.3 Data Types and Cryptographic Primitives

We expect that no more than 2^{32} different building IDs will be required (this enables ample room for changing/restructuring of building Ids). We also expect that no more than 2^{128} token IDs will be required (this enables to if necessary issue multiple token IDs

in case a token is malfunctioning or gets lost).

Therefore a building ID is defined as unsigned 32 bit integer `uint32`, and a token ID is defined via four unsigned 32 bit integers `[uint32, uint32, uint32, uint32]`.

A ticket is a collection of all the building IDs that a user has access to. It is defined as concatenation of 32 bit integers `ticket = buildingID1 buildingID2 ... buildingIDn`.

To authenticate a token it stores its token ID encrypted under a 128 bit system master key (there may be several master keys): `secureID = [uint32, uint32, uint32, uint32]`, `smk = [uint32, uint32, uint32, uint32]`.

A ticket may be held briefly as plaintext when a token is operating, but otherwise it is always kept encrypted under a master key: `secureTicket = Enc(ticket)`.

We will use AES with 128 bit keys as cryptographic primitive. Because all core data is likely to be smaller than 128 bits, we will use AES in ECB mode.

We follow RFC 3526 `https://tools.ietf.org/html/rfc3526` and choose the DH parameters as follows.

```
The prime is: 2^1536 - 2^1472 - 1 + 2^64 * { [2^1406 pi] + 741804 }

Its hexadecimal value is:

FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

The generator is 3. This choice ensures that the DH difficulty is aligned with the AES-128.

To convert the outcome of a DH key exchange to a symmetric session key we will use AES as block cipher to implement CBC-MAC with fixed key equal to zero. This takes the 1536 bit outcome of DH and produces a 128 bit AES session key.

2.4 Token-Reader communication

There are three interactions foreseen between a SuperSAM token and a reader. Any interaction is based on respective lower-level standardised communication. We describe here only the cryptographic functionality of our system.

Authenticate Any interaction needs to be initialised by an authentication. During authentication, the token proves that it is legitimate, and token and reader agree on a secure session key for the rest of the interaction. We achieve this by a Diffie Hellman Key Exchange; where token and reader both supply randomness to agree on a shared session key.

The interaction starts by the Reader initiating, and then the SuperSAM Token offering its `secureID` to the reader, the DH ingredients are being offered alongside.

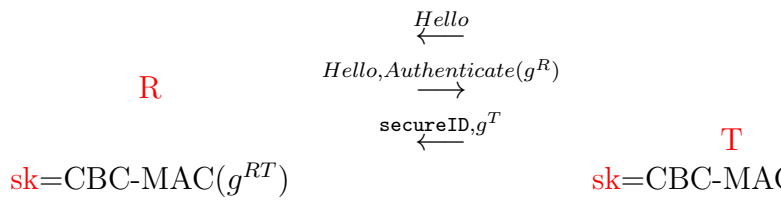
Send Ticket This interaction is only enabled after a successful authentication. The SuperSAM token then sends its `ticket` encrypted under the newly established session key over the the reader.

Update Ticket This interaction is only enabled after a successful authentication and on specific master readers. The reader sends a new `ticket` over to the SuperSAM token.

Update System Master Key To ensure key agility there will be the option of replacing a system master key `smk` on a token. The reader sends a new `smk` over to the token alongside with an encrypted ticket. The token then decrypts the encrypted ticket with the new system master key and checks if the ticket is the same as the previously held ticket. If so (implicit reader authentication), it accepts the new `smk`.

2.4.1 Communication flow

SuperSAM Token is used to gain building access



A Token entering the proximity of the reader announces itself. The Reader requests the token to authenticate and offers its DH value. The token responds by sending its `secureID` and its DH value. If the Reader can successfully verify the `secureID` it well then accept the derived sk and initiate further communication with this token. Otherwise it will drop the connection.

SuperSAM Ticket is updated

Reader $\xrightarrow{UpdateTicket(Enc_{sk}(\texttt{newTicket}))}$ Token

$\texttt{Ticket} = Dec_{sk}(Enc_{sk}(\texttt{newTicket}))$
$\texttt{secureTicket} = Enc_{\texttt{smk}}(\texttt{Ticket}))$

SuperSAM System Master Key is updated

Reader $\xrightarrow{UpdateSMK(Enc_{sk}(\texttt{smk}),\texttt{secureTicket})}$ Token

Decrypt and check

3 Implementation

We have so far selected a prototype implementation of the symmetric cryptographic primitive, AES (128 bit key)[1].

We do not have a prototype implementation of the DH exchange. Subsequently we provide further details for the prototype of the symmetric implementation, and an outline for the DH key exchange implementation strategy.

3.1 Threat Analysis

The SuperSAM token will be given out to an employee therefore we assume that it could be stolen by adversaries. The reader modules will be securely wall mounted, therefore we assume that they will be secure.

Adversaries may want to clone a token, or change the ticket information on the token. They could do that if they can extract an `smk`. Because all communication is super secure, the main attack vector of an adversary will be to analyse the token itself. The strongest attacks in this scenario are side channel and fault attacks. Therefore we need to ensure that the AES implementation in particular is secure against Differential Power Analysis.

3.2 Super Secure AES

The basis of CBC-MAC and Encryption is an implementation of AES. Our implementation is provably secure against all DPA style attacks (it is based on a higher order

[1]This implementation is based on the freely available implementation by Frank Hemsworth, which is available from here: `https://github.com/knarfrank/Higher-Order-Masked-AES-128`

masking scheme designed by Rivain and Prouff[2]).

A higher order masking scheme splits the AES state bytes into many shares and then changes the AES functions to work on the shares. In particular we implement a version of `SecExp254` for SubBytes.

Because the AES state is represented as many shares, fault attacks cannot have any impact on our implementation and are therefore not a threat. Also SPA style attacks are of no consequence to a SuperSAM Token.

We use a highly secure random number generator on SuperSAM to generate the necessary randomness.

SuperSAM Tokens store tickets in an encrypted fashion. A ticket is only ever for a short time frame in memory, thus there is only minimal risk recovering a ticket from an operational token.

3.3 DH Implementation Considerations

We do not have a working prototype for the DH functionality. Our considerations so far are to implement a Montgomery exponentiation, using a special purpose hardware multiplier. Because we will perform both multiplication and squaring on this multiplier, we expect there to be no difference between multiplications and squaring.

Because storage is very limited, we cannot precompute any values, therefor we aim to implement a simple left-to-right binary exponentiation.

We think that no SPA protection is necessary. Because this DH works on ephemeral values, we do not anticipate that DPA style attacks will be a problem. Also because we have such large primes, DPA style attacks would be extremely difficult. Therefore we do not plan to implement any special mitigation techniques for the DH implementation.

3.4 Testing

We have so far tested our AES implementation for functional correctness. We have also tested the PRNG and it satisfies the serial test; therefore it is clear that the resulting distribution is uniform. The PRNG was taken from the original Github implementation and we believe that it is cryptographically secure because it has a long cycle.
We believe that this implementation puts us in a good position to real EAL Level 4 in FIPS 140-3. Although we have yet to start with side channel testing (but as we use a

[2]http://www.matthieurivain.com/files/ches10.pdf

higher order masking scheme, we expect that no attack can succeed because the literature says that they are provably secure.)