AnonAccess

das Labor
`http://www.das-labor.org`

Daniel Otte
daniel.otte@ruhr-uni-bochum.de

Sören Heisrath
sh@3dots.de

December 3, 2007

Abstract

This paper gives an overview of the AnonAccess-system, which tries to provide access to users which may be known by name, pseudonym or a shared pseudonym, to a given functionality (ex. open a door). The shared pseudonym access feature is tried to be extended and implemented in such a way that it can be claimed to be anonymous.

1 Notations and conventions

$a \leftarrow b$	a is asigned the value of b
$a \oplus b$	a xor b
$a \wedge b$	a bit wise and b
$a \vee b$	a bit wise or b
$a \parallel b$	concatenation of the bit strings a and b
$a_{(base)}$	the constant a is given in base *base* notation, if not specified the base is 10
$H(a)$	is the value of the hash function SHA-256 of message a
$HMAC_{key}(a)$	is the value of the HMAC-SHA256 MAC function of message a and key *key*
bit	a bit is the basic unit of information; it can only have one of two values, which we consider to be 1 and 0
byte	a byte is considered to be a group of eight bits throughout this document
Ki, Mi, Gi	prefixes to units, specifying a multiple of $2^{10} = 1,024$, $2^{20} = 1048,576$ and $2^{30} = 1,073,741,824$; see [1] for reasons
K, M, G	prefixes to units, specifying a multiple of $10^3 = 1,000$, $10^6 = 1,000,000$ and $10^9 = 1,000,000,000$

2 Cryptographic algorithms used

We use the following cryptographic primitives:

- SHA-256 hash function as specified in [2]
- HMAC-SHA256 MAC function as specified in [3]
- Shabea with 16 rounds as data encryption algorithm as specified in appendix B
- a PRNG as specified in appendix A

3 Components

The AnonAccess system is divided in *Terminal-Unit* and *Master-Unit*, additionaly there is a chip-card for each user, which stores the user's authentication data.

3.1 Chip-Card

We use simple memory cards with I^2C-Bus[4] and form factor ID-1 as specified in [5][6]. They are quite cheap (less then 1€ per card) and not secure. Their contents might easily be read or modified, so everyone can read and check what we write on his/her card.

The card contains a so called *AuthBlock* embedded in an ASN.1-BER[7] octal-string object. The *AuthBlock* has the following structure:

name	size	description
UID	2 bytes	index to the *TicketDB*
ticket	32 bytes	ticket containing encrypted time-stamp
r_{key}	32 bytes	random key for r_{ID} decryption
r_{ID}	32 bytes	encrypted user pseudonym
HMAC	32 bytes	$HMAC_{absign\_key}(UID \parallel ticket \parallel r_{key} \parallel r_{ID})$

3.2 Terminal-Unit

The *Terminal-Unit* handles user inputs, displays information and reads and writes the user's card. It is equipped with keypad, display, card reader and a hardware random number generator. It'power is supplied by the *Master-Unit* and it should therefore not be reset even in the case of power failure.

3.3 Master-Unit

The *Master-Unit* keeps the databases, does the authentication and executes the secured action (ex. opens a door).

3.4 Power supply

The power supply is designed to power the *Terminal-Unit* and the *Master-Unit*. It uses an accumulator to work as uninterruptible power supply, so that about 60 hours of operation without external power supply should be possible. Therefore under normal circumstances a reset due to a power failure should not happen.

3.5 Real time clock (RTC)

The real time clock is implemented in software by using one of the microcontroller's timers. A timer interrupt function increments a 64bit value each millisecond (this counter will wrap around in about 584.542.046 years, which should be quite enough for us). Additionally the counter's value is periodically[1] written to the microcontroller's EEPROM and read back after reset. On reset we also add the value $3FFFFF_{(16)}$ to the counter to avoid having the same timestamp for more than one time.

The backup storage is implemented in a ring buffer structure with an additional index byte. The index byte indicates which cell of the ring buffer is to be used. After writing a value to a cell it is read back and checked. If the check fails the index byte is incremented by one and the next cell is used. The EEPROM is specified to be written 100,000 times so one cell may work for 116,508.4 hours which is about 13.29 years. So with a ring buffer of 20 cells, we should be able to operate for about 265.82 years which should be sufficient for most applications (if not the ring buffer could be easily made even larger).

It should be known that the timer value does not necessarily correspond to a linear continuous time line or human time, although the time is monotonic increasing.

[1]the value is backed up every $3FFFFF_{(16)}$ milliseconds which is about every 1.165 hours

3.6 Microcontroller

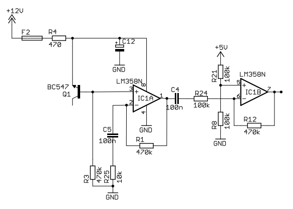
We use microcontrollers from the ATmega family from Atmel[13]for both units. They are relatively cheap and support protection of the internal memories (flash and EEPROM) from being read through their lock-bit feature. There also is a toolchain including GCCs[16] C-compiler and a libc implementation[17] available for these 8 bit microcontrollers which eases the writing of the software.

The *Master-Unit* uses an ATmega644[14] in DIL-Package with 64KiB of program flash, 4KiB of internal SRAM and 2KiB of internal EEPROM (100,000 rewrite cycles guaranteed).

The *Terminal-Unit* uses an ATmega32[15] in DIL-Package with 32KiB of program flash, 2KiB of internal SRAM and 1KiB of internal EEPROM (100,000 rewrite cycles guaranteed).

3.7 Random number generator (RNG)

This circuit utilises the randomness of the transistor diode's breakdown current to generate random voltages in the range from 0 to 5 volts. While this is quite random it does not need to be cryptographically secure, because the RNGs output is used only as input for the cryptographically secure PRNG.



schematic of the hardware random generator

3.8 Pseudo-random number generator (PRNG)

The PRNG is based on the SHA-256 hash function and is specified in appendix A. It has two main functions:

- AddEntropy: this function adds data to the entropy pool, the input can be of arbitrary bit length
- GetRandomBlock: this function fills a 32 byte block of memory with a randomised bit string

Another function (GetRandomByte) uses a buffer and the GetRandomBlock function and returns a random byte. The PRNG is periodically filled with entropy from the hardware RNG using the AddEntropy function.

3.9 Secure serial port (QPort-tiny)

QPort-tiny[11] is a software stack which offers a secure communication channel over an insecure serial line. For that purpose it uses a pre-shared secret key to agree on a set of secret symmetric keys, which are then used for encryption. HMAC-SHA256 is used for session key generation, and XTEA[12] is used in OFB and CFB mode for encryption.

3.10 External serial EEPROM

The external serial EEPROM is used to keep the ticket databases and the flag-modify database, and can be used for key-storage in the migration process.

We use standard I^2C[4] EEPROMs with 512KiBit or 1MiBit (24xx512[8] or 24xx1025[9]) from Microchip[10]. It is possible to extend the storage capabilities by using multiple EEPROMs. That makes it possible to have up to 4MiBit or 512KiBytes of storage space which normally allows more than 10,000 users.

All contents of the EEPROM are encrypted (except the keymigration-area). Shabea-16 is used to encrypt the content. We therefore divide the EEPROM space into 32 byte blocks which are encrypted separately. Every block is encrypted with an individual key which is the result of concatenation of the "main-key"(*eepromcrypt_key*) and the block address. So we are protected from most attacks against mass storage encryption (ex. watermarking).

3.11 Ticket-Database (TicketDB)

This database is used to store a HMAC of the user's ticket, her/his permissions, and some statistics about the whole system. The first element in the database is the header followed by the entries for the users.

Header structure:

name	size	description
ID	10 bytes	set to the ASCII string "AnonAccess"
majversion	1 byte	major version; set to 1
minversion	1 byte	minor version; set to 0
headersize	1 byte	specifies the size of the header
stat	10 bytes	statistics
reserved	8 bytes	reserved field for future extensions and for alignment; set to 0

The statistics field has the following structure:

name	size	description
max_users	2 bytes	maximum number of users
users	2 bytes	actually active user
admins	2 bytes	actually active admins
locked_users	2 bytes	number of locked users
locked_admins	2 bytes	number of locked admins

The following space of the *TicketDB* is filled with user entries which have the following structure:

name	size	description
flags	1 byte	the flags associated with the user
nickname	7 bytes	the nickname if the user decided to be known by name
ticketmac	32 bytes	HMAC from users ticket

Where the flag field has the following structure:

name	size	description
exists	1 bit	indicates if this entry is used (1: in use; 0: free)
admin	1 bit	set if user has admin privileges, cleared otherwise
locked	1 bit	set if user is locked; cleared otherwise
notify_lostadmin	1 bit	set if user has to be notified about lost admin privileges
anonymous	1 bit	set if the user did not specify user name to be stored
reserved	3 bit	reserved, should be set to 0

3.12 FlagModifying-Database (FLMDB)

The flag-modifying-Database keeps entries which specify how a given user account should be modified.

name	size	description
active	1 byte	set to 1 if this entry is active; set to 0 otherwise
permanent	1 byte	set to 1 if this entry should not be removed if applied; set to 0 otherwise
last	1 bytes	if set to 1 this is the last entry to check; set to 0 otherwise
setflags	1 byte	specifies which bits have to be set in the userflags
clearflags	1 byte	specifies which bits have to be cleared in the userflags
reserved	3 byte	reserved; set to 0
timestamp	8 bytes	timestamp of the creation of this entry
hnick	32 bytes	HMAC of the *user pseudonym*

3.13 Key-Database (Key-DB)

This database stores all the cryptographic keys used in the system.

name	size	description
ticket_key	256 bit	used to generate the HMAC from the ticket which is stored in *TicketDB*
absign_key	256 bit	used to generate the HMAC in the *AuthBlock*
rid_key	256 bit	used to encrypt the *user pseudonym*
nick_key	256 bit	used to generate the HMAC from the user's nickname giving the *user pseudonym*
timestamp_key	256 bit	used to generate a new ticket by encrypting a 24 byte random string and a 8 byte timestamp
eepromcrypt_key	256 bit	used for encrypting the external EEPROM's content

4 Being known by name or shared pseudonym

AnonAccess allows three ways of being known:

- being known by name
- being known by pseudonym
- being known by a shared pseudonym

4.1 Being known by name

If the user selects to be known by name the nickname is stored in the *TicketDB* in a way that is available in plaintext to the *Master-Unit*. It can be searched for and it can be read by an administrator. This allows immediate manipulation of the user's flags.

4.2 Being known by pseudonym

In every mode the user enters his/her nickname at card creation time at the *Terminal-Unit*, and the *Master-Unit* generates a HMAC (with a special key, the *nickkey*) from this nickname. This HMAC is referred to as *user pseudonym* in this document. It is neither possible for the *Master-Unit* nor the *Terminal-Unit* to compute the user's nickname from this pseudonym. The *user pseudonym* is not stored in the *Master-Unit* neither in the *Terminal-Unit*, it is stored only in double encrypted form in the *AuthBlock* on the users card.

This pseudonym is used to apply modifications to a given account. A modification is done by adding an entry to the *FLMDB*. As this requires the *user pseudonym*, the nickname of the associated user must be known. Also the modifications can only be applied when the user processes the user authentication process.

4.3 Sharing a pseudonym

It is also possible to have multiple users sharing the same *user pseudonym*. Therefore they simply have to enter the same nickname. It is recommended to use the name of colors for such groups.

To apply modifications to an account in such a group, the modification has to be applied to all members of the group. An exception is the case where the card related to this account is available. In this case the *UID* from the card can be used to modify the flags in the *TicketDB* directly.

5 Usage

This section describes the AnonAccess system from the user's point of view.

5.1 Actions and commands

5.1.1 mainopen

Execute a special action (ex. open a door).

5.1.2 mainclose

Execute a special action (ex. closing/locking a door).

5.1.3 adduser

Add a user to the system. A user nickname must be specified. A user is added by generating a new valid *AuthBlock* which is written to an empty card, and by writing corresponding information to the *TicketDB*.

5.1.4 remuser

Remove a user from the system. A user nickname must be specified. If the nickname is stored in the *TicketDB* the entry in the *TicketDB* is immediately deleted which includes setting the *exists*-flag to 0. If the nickname is not stored in *TicketDB* a new entry in *FLMDB* is generated which leads to removal of the account when a *AuthBlock* is processed whichs *user pseudonym* matches the generated *user pseudonym*.

Table 1: example for minimum permission levels for different tasks

action	requirements
mainopen	1 user
mainclose	1 user
adduser	1 admin
remuser	1 admin
lockuser	1 admin
unlockuser	1 admin
addadmin	2 admins
remadmin	2 admins
keymigrate	3 admins

5.1.5 lockuser

Same as removing a user but instead of deleting the entry only the lock bit is set, which will cause the system to not accept the card as valid user card.

5.1.6 unlockuser

Same as removing a user, but instead of deleting the entry, an eventually set lock bit will be cleared.

5.1.7 addadmin

Same as removing a user, but instead of deleting the entry, the admin bit will be set, granting admin privileges to the user.

5.1.8 remadmin

Same as removing a user, but instead of deleting the entry, an eventually set admin bit will be cleared, so the user will not have admin privileges any more.

5.1.9 keymigrate

Initiate a key-migration, which will write the internal secret keys to the external serial EEPROM. This might not be implemented for security reasons.

5.2 Privileges

The system differentiates between "normal" (non-admin) users and admin users. To execute a given task in a session, special authorisation requirements must be met. These requirements are given as the number of users and admins which have to participate in the session. It might be decided to restrict admin privileges to users which are known by nickname. The given example of minimum permission levels assumes that admin privileges are restricted to users that are known by nickname.

6 Ideal run

1. User inserts card in *Terminal-Unit*

2. *Terminal-Unit* reads *AuthBlock* from card and transmits it in *addAuth*-Packet to *Master-Unit*
3. *Master-Unit* checks *UID* to be in range
4. *Master-Unit* checks *ticket* against the HMAC in *TicketDB* at *UID*
5. *Master-Unit* loads *userflags* from *TicketDB*
6. *Master-Unit* decrypts *ticket* and checks *timestamp* to be in range
7. *Master-Unit* decrypts r_{ID} ($dec_{pseudokey}(dec_{r_{key}}(r_{ID}))$) to get users pseudonym
8. *Master-Unit* searches in *FLMDB* for entries matching users pseudonym; for every matching entry it does:
   (a) modify users flags as indicated by the *setflags* and *clearflags* fields
   (b) delete the entry if the *permanent*-flag is not set
9. *Master-Unit* deletes *TicketDB*-entry
10. *Master-Unit* generates a new *UID* which points to an entry in *TicketDB*
11. *Master-Unit* generates a new *ticket* with a new *timestamp*
12. *Master-Unit* writes new *ticket* at *UID* in *TicketDB*
13. *Master-Unit* generates new r_{key}
14. *Master-Unit* generates new $r_{ID}= enc_{rid\_key}(enc_{r_{key}}(userspseudonym))$
15. *Master-Unit* transmits new *AuthBlock* in *addAuthAck*-Packet to *Terminal-Unit*
16. *Terminal-Unit* writes new *AuthBlock* onto card

7 Attacks and trusted components

This section tries to give an overview of the trust level of components and thereby an overview of the trust level of a complete implementation of AnonAccess.

7.1 Security goals

- access should only be granted to users who have a valid card whichs information and related information in the database state, that access should be granted to this user.
- no valuable information should be retrievable from the card's contents
- no valuable information should be retrievable by an unauthorised user from the AnonAccess system
- no information about the presence of a user who is not known by nickname should be available, even to an user with admin privileges

7.2 Trusted components

We consider a component to be a trusted component if the compromisation of this component leads to compromisation of at least one of the former declared security goals.

7.2.1 Terminal-Unit

The *Terminal-Unit* is considered trusted, especially the connection between the microcontroller and the card must be protected.

7.2.2 Master-Unit

The *Master-Unit* is considered trusted, especially the serial bus between the microcontroller and the external serial EEPROM must be protected. Although the external EEPROM's content is encrypted, an attacker might gather usefull information from the addresses which are accessed.

A The PRNG

The PRNG utilises SHA-256 as hash function. The entropy pool is 64 bytes (512 bits) large, which is the block size of SHA-256. We specify two algorithms which implement the functionality of the PRNG, one to add entropy to the entropy pool and one to get a block (32 bytes) of random data.

Algorithm 1 Add some data to the entropy pool

```
Require: $pool = pool_0 \parallel pool_1$ where $pool_0$ and $pool_1$ are both 32 bytes large
Require: $data$ of arbitrary length
Require: $offset$ which may be 0 or 1
  $temp \leftarrow H(pool \parallel data)$
  $pool_{offset} \leftarrow pool_{offset} \oplus temp$
  $offset \leftarrow offset \oplus 1$
```

Algorithm 2 Get a block of random data from the entropy pool

```
Require: $pool = pool_0 \parallel pool_1$ where $pool_0$ and $pool_1$ are both 32 bytes large
Require: $offset$ which may be 0 or 1
  $temp \leftarrow H(pool)$
  $pool_{offset} \leftarrow pool_{offset} \oplus temp$
  $offset \leftarrow offset \oplus 1$
  $temp[temp[0] \wedge 31] \leftarrow temp[temp[0] \wedge 31] + 1$
  $OUTPUT \leftarrow H(temp)$
```

B the Shabea-Cipher

Shabea (SHA based encryption algorithm) is a SHA-256 based Feistel-Cipher. It was designed to securely encrypt data where a SHA-256 implementation is available. It was important to have a small (in program space and memory

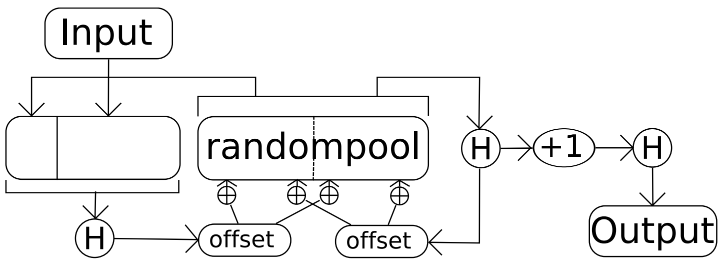


Figure 1: schematic of the PRNG

requirement) and nevertheless secure symmetric cipher, in the case that a SHA-256 implementation is available.

Algorithm 3 Encryption with Shabea

```
Require: $INPUT = L_0 \parallel R_0$ where $L_0$ and $R_0$ are both 16 bytes large
Require: $4 \leq rounds \leq 255$
Require: $key$ which length (in bits) is $keylength$ of any size
  for $i = 0$ to $rounds$ do
    $L_{i+1} \leftarrow R_i$
    $R_{i+1} \leftarrow L_i \oplus H(key \parallel 0 \parallel i \parallel R_i)$
  end for
  $OUTPUT = L_{i+1} \parallel R_{i+1}$
```

Algorithm 4 Decryption with Shabea

```
Require: $INPUT = L_{rounds} \parallel R_{rounds}$ where $L_{rounds}$ and $R_{rounds}$ are both
  16 bytes large
Require: $4 \leq rounds \leq 255$
Require: $key$ which length (in bits) is $keylength$ of any size
  for $i = rounds + 1$ downto 1 do
    $R_{i-1} \leftarrow L_i$
    $L_{i-1} \leftarrow R_i \oplus H(key \parallel 0 \parallel i \parallel L_i)$
  end for
  $OUTPUT = L_0 \parallel R_0$
```

References

[1] When is a kilobyte a kibibyte? And an MB an MiB? (http://www.iec.ch/zone/si/si_bytes.htm)

[2] FIPS 180-2: Secure Hash Standard (SHS) (http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf)

[3] RFC 2104: HMAC: Keyed-Hashing for Message Authentication

[4] The I^2C-Bus Specification, Version 2.1, January 2000, original specification from NXP Semiconductors (http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf)

[5] ISO/IEC 7816-1:1998 Identification cards – Integrated circuit(s) cards with contacts – Part 1: Physical characteristics

[6] ISO/IEC 7816-2:1999 Identification cards – Integrated circuit cards – Part 2: Cards with contacts – Dimensions and location of the contacts

[7] ITU-T Rec. X.690: Information technology ? Abstract Syntax Notation One (ASN.1): Specification of basic notation (http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf)

[8] 24AA512/24LC512/24FC512 1024K I^2C CMOS Serial EEPROM, datasheet by Microchip (http://ww1.microchip.com/downloads/en/DeviceDoc/21754H.pdf)

[9] 24AA1025/24LC1025/24FC1025 1024K I^2C CMOS Serial EEPROM, datasheet by Microchip (http://ww1.microchip.com/downloads/en/DeviceDoc/21941E.pdf)

[10] The Microchip Cooperation web presence (http://www.microchip.com)

[11] QPort-tiny specification, Daniel Otte (http://nerilex.3dots.de/qport-tiny.pdf).

[12] Tea extensions, Roger M. Needham and David J. Wheeler, (Notes October 1996, Revised March 1997, Corrected October 1997) (http://www.cix.co.uk/~klockstone/xtea.pdf)

[13] The Atmel Cooperation web presence (http://www.atmel.com)

[14] ATmega644 Preliminary (revision M, updated 08/07) (http://www.atmel.com/dyn/resources/prod_documents/doc2593.pdf)

[15] ATmega32(L) (revision K, updated 08/07) (http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf)

[16] GCC, the GNU Compiler Collection (http://gcc.gnu.org)

[17] AVR Libc Home Page (http://www.nongnu.org/avr-libc/)