DESIGN AND IMPLEMENTATION OF AN AUTHENTICATION PROTOCOL FOR HOME AUTOMATION SYSTEMS

S. S. Leong and Nicholas C. H. Vun[1]
[1]Nanyang Technological University
School of Applied Science
Nanyang Avenue
Singapore 639798
Email: aschvun@ntu.edu.sg

Abstract

This paper presents an authentication protocol using the SPEKE strong password method to address current information-security deficiencies found in most Smart Home Automation systems. The protocol emphasizes on analyzing operation and security performances limited by practical low-cost micro-controller implementations (e.g. 8051 or 68HC11) of consumer electronic equipment and appliances.

1 Introduction

The push of technology often becomes a double-edged sword when products frequently compromise on information security while rushing to capture market share at minimum cost and confusion to consumers. In focus are rising concerns over the lack of authentication facilities in most Smart Home Automation (HA) standards. One such HA specification is the Consumer Electronic Bus (CEBus®) formally known as EIA-600 standard [EIA95]. As indicated in the specification, EIA-600 is not intended for secure communications and there is no attempt to protect the network from a malicious application/user. Therefore, any desired protection must be provided by other means to control access to a medium or spectrum used by the medium. An ideal solution is to provide signal confinement of CEBus packets within trusted compound [Vun97]. However, such method is costly, non-foolproof and most likely requires alterations to the CEBus specifications. Clearly, modern cryptographic methods are needed to address this problem. The proposed Home Automation Authentication Protocol (HAAP) uses industrial-strength SPEKE [Jab96] public-key exchange for session key generation; MD5 message digest for digital signing of messages; IDEA block cipher for encryption; sequence number challenge-response for added security; dynamic N-time password to defeat dictionary attacks as well as denial of services to mitigate resource flooding attacks.

1.1 Objectives & Design Principles

The ultimate goal is to ensure three information security mechanisms in all CEBus devices. These mechanisms are - privacy, authentication and integrity. In addition, the goal must meet the following added criteria:

1. *Cheap.* The system should not require the user to purchase additional equipment or pay additional monthly fees for authentication services.
2. *Open Standard.* For maximum availability and interoperability, the authentication algorithm should be completely made known to the public without compromising security.

1.2 Document Organization

This paper is organized as follows. Section 2 provides an overview of the fallacies of present suggested CEBus authentication schemes; Section 3 covers the proposed authentication protocol; Section 4 analyses the protocol performance; Section 5 discusses the integration of the protocol into CEBus. Lastly, section 6 mentions some known limitations and assumptions. This document assumes familiarity with the Diffie-Hellman (DH) public-key algorithm [SchBr1]. Further references of this protocol can be found in [LSS98].

1.3 Terminology

Term	Meaning
Initiator, Requester, A	The key establishment party which sends the first CEBus session request (usually referring to the CEBus controller).
Responder, Provider, B	The key establishment party which receives the first CEBus session (usually referring to the CEBus peripheral/device).
S	A shared password for the Initiator and Responder (8-128 bit).
r_B	A randomly generated, prior-hashed new password from Responder to Initiator (≥8 bit).
p	A huge prime number suitable for Diffie-Hellman exchange (≥128 bit).
q	A large prime factor of p-1 (≥128 bit).
R_A, R_B	Random numbers chosen by Initiator, and Responder (8-bit).
Public-value, Q_A, Q_B	The publicly distributed exponential values sent by Initiator and Responder and used in SPEKE to generate K.
E(k, m)	A symmetric encryption function of m using key k (IDEA).
h(m)	A strong one-way hash function of m (MD5).
$A \rightarrow B : m$	Initiator sends m to Responder.
K	The calculated SPEKE session key.
Node	CEBus device/peripheral.

Manuscript received June 17, 1998

0098 3063/98 $10.00 © 1998 IEEE

2 Deficiencies In Current Authentication Methods

2.1 CEBus Threat Model

CEBus is a broadcast network consisting of seven different media. Services across different media are made possible through brouters/bridges. Similar to common Ethernet networks, CEBus is opened to passive tapping and on-line/off-line attacks. Some basic security threats include node masquerading activities, replay attacks, interception of data, manipulation of messages and repudiation of services. Four undesirable CEBus connections are identified in [Vun97]. In order to identify uniquely any valid equipment, it is inevitable to allocate passwords to every trusted node. The only problem remaining now lies in protecting the password during authentication.

In literature [EIA95], two CEBus authentication methods are proposed. We will examine these methods further and explain why they may be unsuitable for home automation systems.

2.2 Challenge-Response Authentication

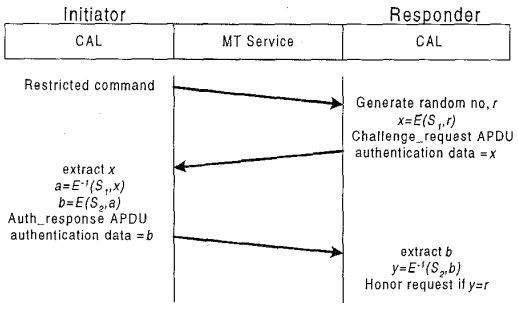


Figure 1 Handshaking for Challenge Response Authentication

To avoid revealing passwords directly, a possible way for the service provider to authenticate the requester is to challenge the requester each time with some prior agreed data responses. The requester would then compute a response. Thus, even if an adversary captures a valid challenge-response pair, it will not help him gain access to the system in the future. With insecure CEBus networks, challenge-response authentication systems are still potential victims to network-based dictionary attack [SRP97]. To illustrate, we analyse the simple protocol in Figure 1, in which an initiator needs to convince another party, the responder, that he knows password S_1. Vice versa, to achieve dual party authentication, the responder needs to know password S_2. We assume that E and E^{-1} are known to the public. Meanwhile, a third party, the adversary, snoops the successful transactions. He can verify a guess S_1' by computing $E(S_1',r)$ using all possible values of S_1 and r and comparing the result against the captured value of x. If they match, he has found the correct password and can now impersonate the initiator.

Here, the adversary utilises a dictionary attack to break S_1. It is effective because in microcontroller based HA devices and appliances, passwords usually are configured by setting DIP switches for practical reason. For human usability, DIP switches are usually 4 to 32 bits wide, which unfortunately narrows the domain of S_1, attributing it susceptible to small-password-space attacks. Assuming E to be common block ciphers such as DES, IDEA or even a simple XOR function, if x is n-bit wide, we can easily deduce r to be also n-bit wide. Although such brute force attack has time complexity of $O(n^2)$, because n is small, the system will be unsecured.

A solution is to regularly change S_1 and S_2 such that it is computationally expensive to attack by brute force. However, given that some peripherals are inconveniently located (e.g. ceiling lighting), frequent password changes will be laborious and thus not feasible.

2.3 One-way Implicit Authentication

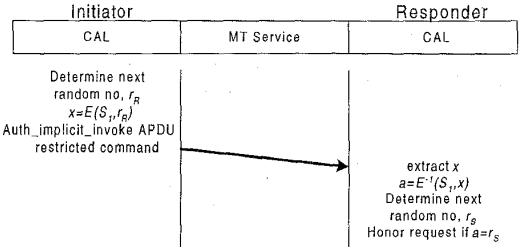


Figure 2 Handshaking for One-way Implicit Authenticiation

Figure 2 shows the protocol for a one-way implicit authentication. Here the ciphertext x is a composite function result containing unknown password, S and random number r_R. Again, a dictionary attack on r_R is possible because it is usually a synchronised changing pseudo random number such as a time stamp [EIA95].

Generation of r_R cannot be too complex because a major objective of CEBus is compatibility [EIA95]. It is intended that every implementation of EIA-600 be able to co-exist with every other implementation regardless of manufacturer. With an adversary regularly recording all messages, and with sufficient ciphertext to guess r_R correctly, knowing r_R would mean that S can also be compromised.

3 Proposed HA Authentication Protocol

Local Mode Home Automation Authentication Protocol (HAAP) is based on public-key cryptography, specifically Strong Password Exponential Key Exchange (SPEKE) [Jab96]. There are four main stages involving both the initiator and the responder. They are: (1) Prime Modulus Establishment, (2) SPEKE Authentication and Session Key Generation, (3) Sequence Number Challenge-Response Authentication and (4) N-Time Password Transformation. In addition, HAAP also defines two counteractive measures to protect against malicious attacks: (A) Defense against

Clogging Attacks and (B) Node Masquerading Prevention. Following subsections will explain HAAP local operation.

3.1 Prime Modulus Establishment

The SPEKE key agreement protocol relies on the difficulty of extracting discrete logarithms modulo a prime and it requires both nodes to agree upon the same prime, p. It is noted that there is no requirement to keep the prime secret or change it frequently. As such, a list of large safe and unique primes can be safely programmed into every appliance permanently. Assuming these primes have widths of 160-bit wide, we can easily store more than 150 primes with just 8KB out of 64KB of ROM space. This prime can be agreed upon in two ways: (1) Programming the prime modulus via human intervention and (2) Negotiating the prime modulus via HA network

3.1.1 Prime Modulus Selection & Programming

A CEBus device cannot operate on its own but needs a corresponding controller (e.g. an IR remote control or a HA-compliant PC) to control it. Before the device is connected to a HA network, the house and unit code and an initial password must be set (normally via DIP switches or jumpers). The next step is to somehow let the controller know the parameters of the new device to use for all future communications. In other words, the service requester(controller) must also choose one of the service provider's prime. This implies that the service requester needs some permanent storage capabilities, which can be inexpensive 8KB EEPROMs that are more than sufficient for such a purpose.

Programming the prime modulus via human intervention. Programming the prime into the remote controller is simple because we can key in our values via the keypad available on the controller. Three parameters are needed: first is the address of the device, next the password and lastly, the prime modulus. The device's address is 32-bit long. But because the keypad entry is in base-10 notation, we will need to key in 10 numbers just for the address code. The password size can be variable, but practical sizes are 8-bit or 16-bit wide (3 to 5 base-10 digits). For the prime, it can be selected by reading it from the device manufacturer's sticker on the device. As mentioned earlier, the appliance may have a list of possible safe primes. There is no preference in the choice of which prime modulus to use, it can be chosen randomly. The same applies if the controller is a computer. However, if a keypad is not available, an alternative solution is to download any one of the service requester's prime to the controller via a physical serial link. This is feasible because most microcontrollers have a built-in serial port.

Negotiating the prime modulus via HA network. For controller devices without direct data entry capabilities such as a simple PLBus On/Off Light Switch Controller, the prime modulus can be agreed via the HA broadcast medium as follows.

PA1. Initiator sends:
x = AUTH_HAAP_PRIME, A→B: x
PA2. Responder sends:
p = AUTH_HAAP_PRIME, B→A: p
PA3. Initiator verifies:
(*ubint_isprime*(p) == 1)?

In prime agreement step PA1, the initiator sets up a byte variable (BV) APDU authentication package (See section 5.2 for more details) to the service provider, requesting for its prime modulus. For higher security, the service provider can choose to send any prime modulus that has not been used before. But this requires additional memory storage to keep track of such status. After p has been sent, the initiator can verify the validity of the received prime by passing it through a stringent Rabin-Miller primality test in step PA3. If the test fails, the service requester will have to repeat the procedure, starting from step PA1, until a safe prime is obtained.

Step PA3 is strictly required because we must be aware that p is sent over an insecure network, hence, we cannot assume p is sent from the intended sender, which could either be physically down or momentarily servicing another node. Such an attack scenario can happen if a malicious node, typically with faster processing elements, sends an unsafe prime to the initiator quicker than the service provider does. If p is unsafe or "cooked", this could lead to leakage of the requester's shared secret to a malicious user.

3.1.2 Prime Modulus Verification

The common prime, once settled, can be used for all future exchanges. *h*() is a one-way hash using MD5. The prime modulus selection works as follows:

PS1. Initiator computes:
HHP = *h*(*h*(p)), A→B: HHP
PS2. Responder checks:
(HHP==LUTHHP(i))? HP = LUTHP(i), B→A: HP
PS3. Initiator verifies:
(HP == *h*(p))?

When an initiator wants to request a service, it must tell the service provider which prime modulus it possesses. This can be done by sending a double hashed of the prime modulus agreed earlier as illustrated in prime selection step PS1. In PS2, the responder will check the received HHP against its list of HHPs with the help of a lookup table (LUTHHP). A lookup table is advisable because performing real-time hashing on a list of prime modulus is computationally wasteful. If a match is found, it sends the corresponding HP, again from another lookup table (LUTHP), to the initiator. HHP is not chosen for the return authentication because a malicious user may replay the previously captured HHP, thus fooling the initiator into a successful prime selection when it may be not. Once the initiator verifies a valid HP, they have successfully chosen a prime modulus for the session.

3.1.3 Advantages Of Pseudo-secret Prime Selection

If the initial prime modulus agreement is performed over a safe channel (i.e. not over the untrusted HA network), we can largely consider this prime to be "private"[1]. Likewise, it is not desirable to negotiate the prime modulus via the HA network because this ruins the privacy token. We observed that the prime modulus selection only involves sending one-way hash values of the prime, thus if this prime modulus privacy is ensured, it will make the problem of attacking SPEKE even harder.

Next, the main reason for having multiple primes is to provide ample room for security. This ensures that every prime modulus that an appliance contains is only shared within a small number of nodes, most favorably one. Conversely, if there are only few primes, when a node requests a once "private" prime over the network, the prime is immediately made public and we have to depend on SPEKE and the others for security.

3.2 SPEKE Authentication & Session Key Generation

In SPEKE, both the initiator and responder will need to use the shared password S[2] to determine the parameters for the DH protocol. $h(S)$ is a secure one-way hash of password S using MD5 [Riv92]. R_A and R_B are two numbers chosen at random by both parties. Finally, HAAP uses prime p in Z_p^* where $p = 2q+1$ for a large prime q. The 2 in the exponent forces the exponential to be a generator of the subgroup of order q. [Jab96] explains this further. Selection of SPEKE generator values will be covered in section 4.2. Steps S1 to S4 is similar to a DH exchange:

S1. Initiator computes:
$Q_A = h(S)^{2R_A} \ mod\ p$, A→B: Q_A
S2. Responder computes:
$Q_B = h(S)^{2R_B} \ mod\ p$, B→A: Q_B
S3a. Initiator computes:
$K_A = Q_B^{2R_A} \ mod\ p = h(S)^{4R_AR_B} \ mod\ p$,
S3b. $HK_A = h(K_A)$,
S3c. $HHK_A = h(HK_A)$, A→B: HHK_A
S4a. Responder computes:
$K_B = Q_A^{2R_B} \ mod\ p = h(S)^{4R_AR_B} \ mod\ p$,
S4b. $HK_B = h(K_B)$,
S4c. $HHK_B = h(HK_B)$,
S5. Responder checks:
$(HHK_A' == HHK_B')$ && $(HHK_B != h(h(1)))$?

In the verification step S5, both parties will compute a hash of their calculated K_X values where the initiator's HHK_A is checked against the responder's HHK_B. This is a slight variation over the original SPEKE that requires both parties to participate in the verification process. The rationale to ensure that only the initiator shows his knowledge of HHK is because HHK_B is a function result that contains the responder's valid password S. But we are not sure of HHK_A. Although SPEKE is immune to the revelation of S, in the event of a discrete log computation, we can be sure that HHK_B is not available for any attacks.

If step S5 fails or HHK_A equals to $h(h(1))$, the responder will not terminate the session until it has received encrypted double hashed sequence number, $EHHSN_A$ from the initiator or when the MT Challenge Timer expires. The purpose is to ensure minimal information is leaked to an attacker where the authentication fails. On the other hand, if Step S5 passes, the initiator is deemed authenticated and HK_A and HK_B will be used as the session key.

Most sub-keys in SPEKE can be pre-computed, and all keys must be destroyed immediately after use, so that they cannot be recovered. In HAAP, the generated session key has a limited lifetime of 8 seconds. The most important reason is protection against cryptanalysis. Another reason for limiting the lifetime of this key is to minimise access to the appliance from a compromised key.

3.3 Extending HAAP With Sequence Number Challenge-Response Authentication

In section 2.3, the security is convoluted by the inclusion of a random variable r_R. Let us now consider the notion of including such facility into HAAP. We begin by illustrating the protocol exchange immediately after a successful SPEKE authentication. Here SN_A refers to the access frequency of the initiator to the responder services; Similarly, SN_B refers to the number of times the initiator has accessed the servicing node. E is an implementation of IDEA [LMM92] 64 bit iterative block cipher algorithm.

A1a. Initiator computes:
$EHHSN_A = E(HK_A, h(h(SN_A)))$, A→B: $EHHSN_A$
A1b. $EHSN_A = E(HK_A, h(SN_A))$,
A2a. Responder computes:
$EHHSN_B = E(HK_B, h(h(SN_B)))$,
A2b. $EHSN_B = E(HK_B, h(SN_B))$,
A3. Responder checks:
$(EHHSN_A == EHHSN_B)$? B→A: $EHSN_B$
A4. Initiator checks:
$(EHSN_A == EHSN_B)$?
A5. If so, Initiator stores:
$SN_A = SN_A + 1$
A6. If so, Responder stores:
$SN_B = SN_B + 1$

In activity steps 1 and 2, both parties will encrypt their known copy of hashed and double hashed sequence number with the session key just derived from the SPEKE operation. The initiator will need to send $EHHSN_A$ to the servicing node where the two 128 bit numbers[3] are verified. If there is a match, the responder replies with $EHSN_B$. Both parties

[1] If private primes are unknown to an adversary, computing discrete logarithms on SPEKE is computationally infeasible

[2] Password S is assumed to be derived from DIP switches/jumper settings on the HA microcontroller board

[3] MD5 output is 128 bits wide; IDEA ciphertext width is similar to its plaintext equivalent

will advance their stored sequence numbers upon a successful verification.

If the authentication fails either here or at SPEKE, an AUTHICATE_REJECT APDU will be sent to the initiator after this authentication. In the light of network security, this AUTHICATE_REJECT APDU in fact has very little meaning. This is because another node can masquerade as the service provider and sends one such packet. Contradictorily, we cannot send a safe AUTHICATE_REJECT packet to the initiator because a secure channel cannot be setup. However to abide to the EIA600 CEBus standard, this is done.

Using sequence number challenge-response authentication has its merit of further ensuring that the initiator knows some past information about their previous communications. Conversely, an adversary who taps an insecure bus sporadically will not be able to synchronise perfectly with the sequence number and thus access will be denied. Stream cipher is another alternative but sequence number challenge-response is preferred due to its simplicity and relatively low CPU utilisation, which makes it suitable to be implemented on low-cost microcontrollers. Of course, many other methods are available and are mentioned in [RSA96].

In section 2.3, the basic aim of an attack is to find S by performing a dictionary attack on S' and r_R. Here, it would be meaningless to find the key HK_B since the session key always changes in the next communication. Instead, an adversary will be more interested to gain knowledge of SN_B, since it is an essential key in the authentication process. Nevertheless, efforts will not be simple because of three reasons. First, HK_B and SN_B are both non-constant parameters. Second, the responder reveals no information on SN_B; an attacker could only trial on error. Lastly, $h()$ (MD5) and E() (IDEA) makes attacks computationally infeasible.

To keep production costs low, some microcontroller systems may not include non-volatile storage resources such as NVRAM. On the event of a power outage, all essential persistent data will be lost. With this assumption, HAAP has to include a sequence number status flag[4] into its protocol stack so that both the initiator and responder can indicate such events to each other. See section 5.2 on the Authentication APDU format. There are four possible synchronisation states:

T1. Both the service requester and provider have no prior communication between them. Indicates a total house power outage or newly purchased equipment.

T2. Service provider has communicated with requester before but requester suffered an initialisation and lost all prior sequence number knowledge. Alternatively, it can also mean an adversary masquerading as a valid requester attempting to bypass sequence number authentication by pretending to have lost its information.

T3. Service requester has communicated with provider before but provider suffered an initialisation and lost all prior sequence number knowledge. Alternatively, it can also mean an adversary masquerading as a valid provider attempting to bypass sequence number authentication in an active attack.

T4. Service provider has communicated with requester before and both are ready to authenticate as described in activity steps A1 to A6.

For authentic information lost in valid nodes, it is up to the system designer to act appropriately. HAAP suggests restricting resource allocation to sensitive devices (e.g. security operations) but allowing other devices (e.g. Lighting/HVAC) to operate normally. Once the synchronisation mismatch is reasoned out, the provider/requester can initialise their sequence number to a fixed initial value via a press switch on the device. For security risk-free systems, the sequence number can automatically be initialised to zero via software once SNFlag is zero. On a final note, it is still advisable to use permanent storage (e.g. cheap EEPROM) so that the system is prevented from resetting to its "new-equipment" state.

3.4 Extending HAAP With Dynamic N-time Password Transformation

We have mentioned passwords configured with DIP switches in section 2.2. Because this password S is not cryptographically large, this carries significant security risks. Hence, it must not be used as a long-lived key to initiate all SPEKE authentications. In fact, such a default key should only be used when two nodes are communicating for the first time.

Dynamic N-time password belongs to an authentication class known as inconvenient authentication. N-time password means that the password is valid only for N times and then discarded. The security advantage of N-time password is that any intercepted password S will become obsolete by the time it is reengineered from a SPEKE session. The drawback is that the service provider will need to regularly generate random password for a service requester. It is also implied that both parties have volatile storage to store this password.

Let "|Z|" denotes the number of bits of nonce Z. Assume $|h(S)| = |HK_A| = |HK_B| = k$, $|EP_A| = |EP_B| = j$, "$x||_{\lfloor y \rfloor}$" denotes extracting y-bits of literal x starting from LSB, "$x||^{\lceil y \rceil}$" means extracting y-bits of literal x starting from MSB and "|" refers to string concatenation:

P1. Provider computes:
$EP_B = E(\ HK_B||_{\lfloor k/2 \rfloor} | \ h(S)||_{\lfloor k/2 \rfloor}, h(r_B) \ | \ h(S)||_{\lfloor k/2 \rfloor}\ )$,
$A \rightarrow B$: EP_B

[4] 1 bit SN FLAG in BV APDU

P2. Requester computes:
$EP_A = E^{-1}(HK_A||_{\lfloor k/2 \rfloor} | h(S)||_{\lfloor k/2 \rfloor}, EP_B)$
P3. Requester extracts:
$EP_A' = EP_A||_{\lceil 2j/3 \rceil}$, $EP_A'' = EP_A||_{\lfloor j/3 \rfloor}$
P4. Requester verifies:
$(EP_A'' == h(S)||_{\lfloor k/2 \rfloor})$?
P5. Requester computes:
$EP_A''' = E(h(S)||_{\lfloor k/2 \rfloor} | HK_A||_{\lfloor k/2 \rfloor}, h(S)||_{\lfloor k/2 \rfloor} | h(r_B))$
$B \rightarrow A: EP_A'''$
P6. Requester stores:
$S = EP_A'$
P7. Provider computes:
$EP_B = E^{-1}(h(S)||_{\lfloor k/2 \rfloor} | HK_B||_{\lfloor k/2 \rfloor}, EP_A''')$
P8. Provider extracts:
$EP_B' = EP_B||_{\lceil j/3 \rceil}$, $EP_B'' = EP_B||_{\lfloor 2j/3 \rfloor}$
P9. Provider verifies:
$((EP_B'' == h(r_B)) \&\& (EP_B' == h(S)||_{\lfloor k/2 \rfloor}))$?
P10. Provider stores:
$S = h(r_B)$

HAAP specifies the password generation and distribution procedure in steps P1 to P10. Here, the service provider is the node responsible for any new password generation[5]. It hashes a random number r_B using MD5 and concatenates the lower 64-bit of current $h(S)$ on the right. Next, it again concatenates the lower 64-bit of current $h(S)$ to the right of the lower 64-bit of derived session key HK_B. This will be used as the 128-bit key to encrypt the former data we have formed earlier (see P1). On receipt, the requester will decrypt the data using a key derived from the right concatenation of the lower 64-bit $h(S)$ with the lower 64-bit of current session key HK_A (see P2). The decrypted data is split into two parts – a 128-bit value new password and a 64-bit portion of $h(S)$ (see P3). This decrypted 64-bit EP_A'' is verified with the requester's lower 64-bit of $h(S)$ (see P4). If they match, the new password EP_A' is authentic and will be used for all future references as S between the provider and itself. To acknowledge the service provider about the successful receipt of this password, both the requester and the provider repeat all steps each other has done. Note that the password concatenation is now left instead of right. When the passwords are again validated, the provider will used the $h(r_B)$ for all communications between this requester and itself. When the two nodes communicate again in the future, they are supposed to use the new password.

First, a quick look may show a potential Achilles heel in such a key management protocol – the problem of password chaining. Password chaining is the ability of an attacker to use an old password (herein referring to the SPEKE session key) to determine a new one (S). However, it has been shown in [Jab96] that to retrieve the session key, we first have to find R_B and R_A random exponents with massive discrete log computations. And this remains a very difficult problem since the search for HK_A requires a log computation for each candidate password S_i. If, however, the session key HK_A is somehow compromised, finding S will still take considerable time. Hence, discrete log function actually creates a chicken and egg problem.

Advantageously, we may derive our IDEA encryption with the concatenation of the two keys just as in P1. Thus there is no known way to find $h(r_B)$ unless we have both the session key as well as the password S. We see that $h(S)$ is also concatenated to the new password. The purpose is to enable the initiator to verify the validity of this package after a decryption. If this facility is unavailable, it may potentially decrypt a malicious packet and voiding all future communications with the service provider. The reverse is done to acknowledge the service provider of the successful password updating. We also observed that only the lower 64-bit of $h(S)$ is used, forming a total ciphertext length of 128-bit + 64-bit = 192-bit (24 bytes). This is due to the 29 bytes data limitation each CEBus packet can transport.

With such an implementation, we can be assured that the new password distribution is largely safe. A dictionary attack on EP_B or EP_A will not be probable because the encryption key is cryptographically large (128-bit) and that IDEA is (still) resistant to all known forms of attacks. Nonetheless, dynamic N-time password remains effective against most casual adversaries.

3.5 Extending HAAP To Defend Against Clogging Attacks

Authentication is necessary, but not sufficient. The computing resources themselves must also be protected against malicious attack or sabotage. To grant access to authorised users regardless of location, it must be possible to cheaply detect and discard malicious CEBus requests. Otherwise, an attacker intent on sabotage can rapidly send CEBus packets to exhaust the servicing node's CPU and/or memory resources. This is because SPEKE operations such as modular exponentiation and encryption are very CPU intensive. To ensure that the processor is not overloaded, this resource clogging vulnerability must not be overlooked. This kind of attack is known as Denial-of-Service attacks.

Although a complete defense against such attacks is impossible, careful logging and simple accounting of erroneous attempts make them much more difficult. The method is to monitor the number of void attempts from any particular CEBus node[6]. In HAAP, if a certain threshold is exceeded, intentional delay to resource allocation must be applied to these offending nodes. For new controllers, an authentication session can only start when a prefixed delay expires after the new device first registers itself with the servicing node via any UNIVERSAL object access. This is to discourage unit address spoofing. There are two benefits with such implementations: first, the CPU is freed from any clogging activities and second, it disfavors online attacks.

[5] Password generation is only possible after the node passes the two authentication algorithms as mentioned earlier

[6] Both house code and unit code of the sender are available in the CEBus LPDU packet

3.6 Extending HAAP To Prevent Node Masquerading

Another problem with CEBus is address spoofing or node masquerading activities. In such an attack, the malicious node can impersonate itself both as a service requester as well as a service provider. The attack may launch right at the start of a request, or it may join in the middle of an exchange and finally, it may even initiate its own requests.

An unorthodox and naïve method is to operate the CENode module in the MONITOR mode [Itln94]. In the MONITOR mode, the CENode interprets all valid LPDU regardless of house and unit code for host post-processing. On detection of another node masquerading as itself, it can immediately send an APDU UACK_REJECT or AUTHICATE_REJECT message. The service node could then cancel the previous service whilst the adversary is in the middle of the authentication process. But honouring this transaction would also imply that an adversary could ruin any valid transaction by sending such a REJECT sequence between two legitimate parties. Thus, this method is only feasible if there is great requirement on design simplicity and/or restrictions on code size. Obviously, a better protocol is needed.

In HAAP design, it will honour any valid packet as long as it is expecting the receipt of such a packet. For example, if a service provider is expecting Q_A from the initiator, and if both the valid initiator and an impersonating node sends their Q_A, the service provider will compute and store both HK_B and HHK_B (see S3a – S3c). Upon the receipt of both copies of HHK_A, it will know which set of values to discard. This example illustrates the worse possible situation since exponentiation is a very expensive computation. However, this method guarantees that valid nodes are rightfully serviced and not denied.

As far as possible, HAAP implements sufficient negotiation to prevent such activities. For example, any handshaking involving verification of data are explicitly accomplished with a double hash value together with a returned hash value of the data. If this verification is one way implicitly, security may be compromised. Below highlights some of the measures taken: (1) Prime selection handshaking is performed before a prime is selected; (2) Sequence number handshaking is done before the sequence number is incremented; (3) New password distribution handshaking is implemented before the new password is accepted

3.7 Key Features Of HAAP

To summarise, this section briefly outlines some of the key features of HAAP with characteristics of SPEKE abstracted from [Jab96]:

1. *Prevent offline dictionary attack on small passwords*
   All passwords are vulnerable to dictionary attacks; SPEKE removes this opportunity because the password is never passed over the network.
2. *Immune to online dictionary and replay attack*
   None of the session key or sub-keys exchanged can be re-used once they are expired. Online attacks can also be stopped by counting access failures and delaying resource allocation to offending and new nodes.
3. *Provide mutual authentication*
   SPEKE allows two parties to prove the knowledge of password S mutually without revealing any information about S.
4. *Integrated key exchange*
   SPEKE knits authentication and key exchanges together, thus eliminating possible attacks. At the same time, a shared session key is safely created which can be used for downstream encryption.
5. *Perfect forward secrecy*
   Theft of the private key, S and shared secret key, HK_X used to sign the exchanges would only allow the thief to impersonate the party in future conversations, but it would not decode any past recorded traffic.
6. *Direct authentication*
   SPEKE verifies the initiator before he is considered authenticated.
7. *No timestamps needed*
   Time-dependent data, which might require time synchronisation between parties, are avoided. This also eliminates any issues concerning trusting of the timeserver.
8. *Sequence number authentication*
   Sequence number challenge-response protocol increases robustness of HAAP.
9. *Dynamic N-time Password*
   HAAP changes the password after a pre-defined access frequency to thwart dictionary attacks on password S.

4 Performance & Implementation Issues

Despite the numerous breakthroughs on stronger authentication methods, most algorithms are safe only with theoretically large generator parameters. Practically this is not realisable with low-cost microcontroller designs. In this subsection, we will analyse the constraints of such practical limitations and make realistic estimates of the range of values SPEKE generator values can take.

A practical CEBus device will be a microcontroller based system fabricated with 12MHz 8032 microcontroller, fitted with 64KB EPROMs (27C512), 32KB SRAMs (62256) and connected to a CEBus Power-Line Bus (PLBus) via CENode DLL interface boards from Intellon. The concrete implementation must fulfil as a real-time interactive system [BEN94] with response time not exceeding two seconds.

4.1 HAAP Module Performance Analysis

To benchmark the total throughput HAAP can handle as well as to determine the appropriate hardware, each individual module is benchmarked with various data sizes using 8051 as a reference. Module performances are in the form of timing equations (8051 CPU cycles) for generality so that HAAP performance can be quickly ascertained for different implementation platforms.

4.1.1 Cryptography Algorithms

$T_{MD5String} = k(260/32) + 124{,}233$ (B1)[A]
$T_{idea\_cipher} = k(14{,}348/32) + 13{,}380$ (B2)[A]
$T_{idea\_decipher} = k(14{,}352/32) + 48{,}347$ (B3)[A]

4.1.2 Big Number Arithmetic

$T_{add} \approx 4kt_{add}$ and $T_{sub} \approx 4kt_{sub}$ (C1)[T]
$T_{mul} \approx i \cdot j \cdot T_{ubint\_mul} \approx 64ijt_{add}$ (C2)[T]
$T_{div} \approx 16t_{add}$ (C3)[T]
$(T_{mulmod} - 1) + (T_{mulmod} + 1) = 2(T_{mulmod} - 1)$ (C4a)[Tmax]
$(T_{mulmod} - 1) + 0 = T_{mulmod} - 1$ (C4b)[Tmin]
$(T_{mulmod} - 1) + \frac{1}{2}(T_{mulmod} - 1) = 3/2\,(T_{mulmod} - 1)$ (C4c)[Tavg]
$T_{mulmod} \approx T_{mul} + T_{div} \approx (64k^2 + 16)\,t_{add}$ (C5)[T]
$T_{ubint\_ccp} = T_{ubint\_cmp} \approx k(115/32)$ (C6)[A]
$T_{ubint\_shl} = T_{ubint\_shr} \approx 0.412k^2 + 47k$ (C7)[A]
$T_{ubint\_add} = T_{ubint\_sub} \approx k(844/32)$ (C8)[A]
$T_{ubint\_mul} \approx 4k^2 + 61k + 456$ (C9)[A]
$T_{ubint\_isprime} \approx 3{,}864k^2 + 203{,}759k + 13{,}823{,}392$ (C10)[A]

4.1.3 Random Number Generator

$T_{random\_time\_seed} = 2{,}540$ (D1)[A]
$T_{random\_time\_seed} = 2{,}363{,}556$ (D2)[A]
$T_{random\_rand} = k(374/16)$ (D3)[A]

4.1.4 Miscellaneous Support

$T_{cebus\_transmit} = 1{,}487$ (M1)[A]
$T_{cenode\_pkt\_process} = 2{,}163$ (M2)[A]
$T_{strcmp} = 535$ (M3)[A]
$T_{strcmpLUT[n]} = nT_{strcmp} = 535n$ (M4)[A]

4.2 HAAP Timing Analysis

With known performance characteristics of each individual module, we can now perform a detailed timing analysis on the HAAP algorithm. Given that large integer arithmetic is expensive, together with HAAP secure coverage, response time is envisaged to be long. But, how long is long? At this point, we have to make some assumptions on the size of HAAP parameters to use. Assume:

- $|S| = 128$-bit (N-Time Password = 128-bit; default DIP switch Password could be smaller)
- $|p| = 160$-bit
- $|R_A| = |R_B| = 8$-bit
- $|SN_A| = |SN_B| = 32$-bit
- $|r_B| = 8$-bit
- Number of entries in HP and HHP lookup table = 150
- Some commands take negligible amount of time (e.g. P6 and P10)
- Worst case timings selected for all operations

With reference to Figure 3, the worst case duration for each complete authentication session would take 151,309,486 cycles, or approximately 151 seconds. This is unacceptable for real-time interactive operations. Next, we will examine the possible optimisations that can be implemented to further reduce the computation time. These are:

1. *Caching common parameters and reduce redundant parameters*
   We notice that HHP, HP, $h(S)$, $h(h(S))$, HK_x and HHK_x are often used. If these are computed and stored, it will save many unnecessary re-hashing operations. This will reduce timings in [PS1, PS3, S1, S2, S5, P1, P2, P4, P5, P7, P9, P10]. We also observed that prime modulus agreement over the network is optional because this can be agreed upon during device installation. This will omit [PA1, PA2, PA3].

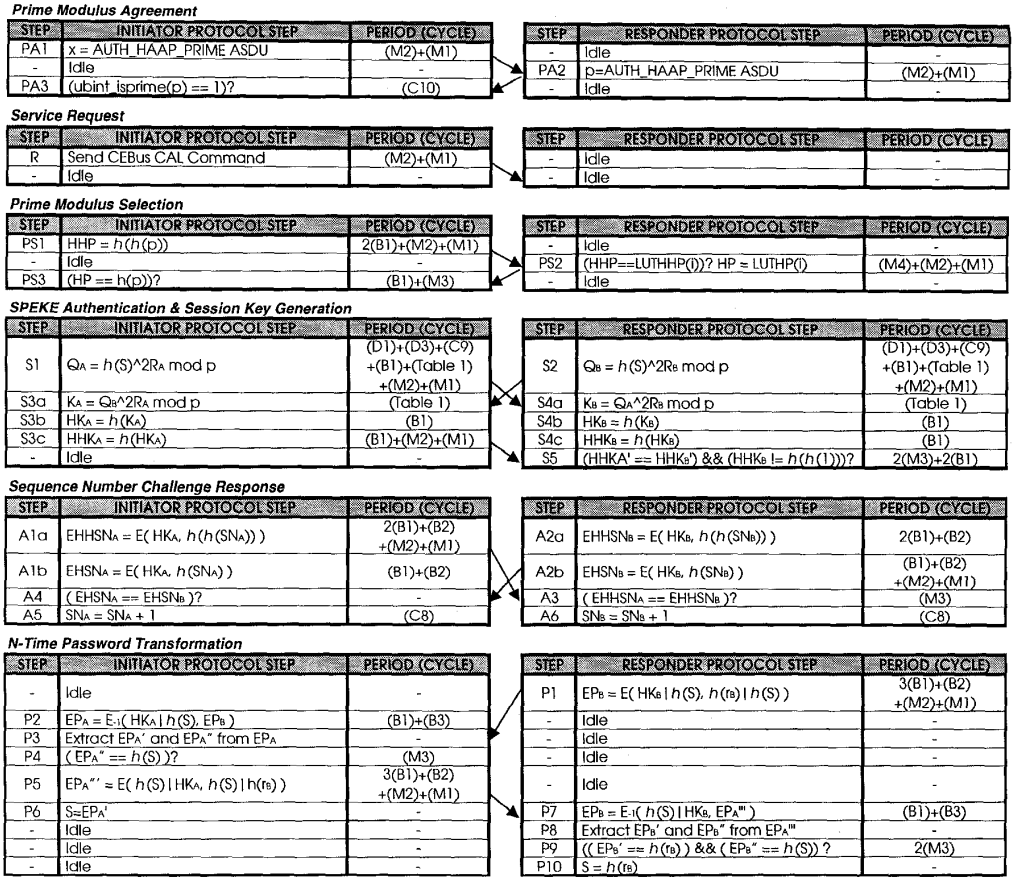
2. *Maximising parallelism*
   The ability to pre-compute any static parameters could reduce overheads significantly. This can be performed during idle time when the device is not authenticating. This can reduce timings in [PS1, PS3, S1, S2, S5, A1a, A1b, A2a, A2b, P1, P2, P4, P5, P6, P9].

   We also note that once Sequence Number Challenge-Response authentication is completed successfully, we can immediate service the CAL request before performing N-Time Password Transformation. This effectively improves the response time greatly. Unless otherwise, this will exclude all timings in [P1, P2, P3, P4, P5, P6, P7, P8, P9, P10].

   Just like in RISC processors, parallelism can also be improved via instruction reordering [Jab97]. Looking closely at steps [A1a, A1b, A2a, A2b], we realised that the initiator's $EHHSN_A$ is not verified immediately by the responder. If we reorder step A3 to execute before step A2b, we can save one MD5 hashing operation.

3. *Reducing size of SPEKE generator size*
   There are several tradeoffs. For example, making the modulus size small increases the chances of discrete log attack. However, this decreases computation time for primality testing. [Jab96] recommends 512-1024 bits for p, 160 bits for random exponents, resulting in an 80 bit K. Here, HAAP uses 160 bits for p and 8 bits for random exponents, which is already many times smaller. Any further reduction in generator size will compromise security and should be avoided.

[A] Actual benchmarked figures (variable *k* and *n* denote bit sizes)
[T] Theoretical figures (variables *k* and *n* denote bit sizes)



Prime Modulus Agreement

STEP	INITIATOR PROTOCOL STEP	PERIOD (CYCLE)
PA1	x = AUTH_HAAP_PRIME ASDU	(M2)+(M1)
-	Idle	-
PA3	(ubint_isprime(p) == 1)?	(C10)

STEP	RESPONDER PROTOCOL STEP	PERIOD (CYCLE)
-	Idle	-
PA2	p=AUTH_HAAP_PRIME ASDU	(M2)+(M1)
-	Idle	-

Service Request

STEP	INITIATOR PROTOCOL STEP	PERIOD (CYCLE)
R	Send CEBus CAL Command	(M2)+(M1)
-	Idle	-

STEP	RESPONDER PROTOCOL STEP	PERIOD (CYCLE)
-	Idle	-
-	Idle	-

Prime Modulus Selection

STEP	INITIATOR PROTOCOL STEP	PERIOD (CYCLE)
PS1	HHP = $h(h(p))$	2(B1)+(M2)+(M1)
-	Idle	-
PS3	(HP == h(p))?	(B1)+(M3)

STEP	RESPONDER PROTOCOL STEP	PERIOD (CYCLE)
-	Idle	-
PS2	(HHP==LUTHHP(I))? HP = LUTHP(I)	(M4)+(M2)+(M1)
-	Idle	-

SPEKE Authentication & Session Key Generation

STEP	INITIATOR PROTOCOL STEP	PERIOD (CYCLE)
S1	$Q_A = h(S)^2R_A \bmod p$	(D1)+(D3)+(C9) +(B1)+(Table 1) +(M2)+(M1)
S3a	$K_A = Q_B^2R_A \bmod p$	(Table 1)
S3b	$HK_A = h(K_A)$	(B1)
S3c	$HHK_A = h(HK_A)$	(B1)+(M2)+(M1)
-	Idle	-

STEP	RESPONDER PROTOCOL STEP	PERIOD (CYCLE)
S2	$Q_B = h(S)^2R_B \bmod p$	(D1)+(D3)+(C9) +(B1)+(Table 1) +(M2)+(M1)
S4a	$K_B = Q_A^2R_B \bmod p$	(Table 1)
S4b	$HK_B = h(K_B)$	(B1)
S4c	$HHK_B = h(HK_B)$	(B1)
S5	(HHKA' == HHKB') && (HHKB != $h(h(1))$)?	2(M3)+2(B1)

Sequence Number Challenge Response

STEP	INITIATOR PROTOCOL STEP	PERIOD (CYCLE)
A1a	$EHHSN_A = E(HK_A, h(h(SN_A)))$	2(B1)+(B2) +(M2)+(M1)
A1b	$EHSN_A = E(HK_A, h(SN_A))$	(B1)+(B2)
A4	($EHSN_A$ == $EHSN_B$)?	-
A5	$SN_A = SN_A + 1$	(C8)

STEP	RESPONDER PROTOCOL STEP	PERIOD (CYCLE)
A2a	$EHHSN_B = E(HK_B, h(h(SN_B)))$	2(B1)+(B2)
A2b	$EHSN_B = E(HK_B, h(SN_B))$	(B1)+(B2) +(M2)+(M1)
A3	($EHHSN_A$ == $EHHSN_B$)?	(M3)
A6	$SN_B = SN_B + 1$	(C8)

N-Time Password Transformation

STEP	INITIATOR PROTOCOL STEP	PERIOD (CYCLE)
-	Idle	-
P2	$EP_A = E_{HK_A \mid h(S)}(EP_B)$	(B1)+(B3)
P3	Extract EP_A' and EP_A'' from EP_A	-
P4	(EP_A'' == $h(S)$)?	(M3)
P5	$EP_A''' = E(h(S) \mid HK_A, h(S) \mid h(r_B))$	3(B1)+(B2) +(M2)+(M1)
P6	$S=EP_A'$	-
-	Idle	-
-	Idle	-
-	Idle	-

STEP	RESPONDER PROTOCOL STEP	PERIOD (CYCLE)
P1	$EP_B = E(HK_B \mid h(S), h(r_B) \mid h(S))$	3(B1)+(B2) +(M2)+(M1)
-	Idle	-
-	Idle	-
-	Idle	-
-	Idle	-
P7	$EP_B = E_{HK_B \mid h(S)}(EP_A''')$	(B1)+(B3)
P8	Extract EP_B' and EP_B'' from EP_A'''	-
P9	((EP_B' == $h(r_B)$) && (EP_B'' == $h(S)$))?	2(M3)
P10	$S = h(r_B)$	-

Figure 3 HAAP Timing Analysis

4. *Reducing traffic*
   HAAP protocol stack specially cater for sending of multiply HAAP parameters within a single CEBus packet. If this is done, we can send [R] and [PS1] together, thus reducing CEBus traffic as well as transmission overheads.

5. *Increasing N for N-Time Password*
   If [P1-P10] cannot be disregarded, we can improve response time by increasing the value of N. Increasing N means that the frequency of password modifications will also reduce by N time. N should not be too big such that the functionality becomes ineffective. HAAP recommends N to be within 2 to 8 for small r_B and much larger N for larger entropies of r_B.

With these optimisations, computation time reduces to [R, PS3*, S1*, S2*, S3a*, S4a*, S5*, A1a*, A1b*, A2a*, A2b*] where "*" denotes "part of". Furthermore, both the service requester and provider can process concurrently, thus further reduction is possible.

The final worst-case response time is thus 1,970,439 cycles or approximately 2 seconds elapsed time, which looks to be pleasantly good. In practical usage, the response time should be lower.

IQI	IRI	Ipi	Cycle Time
8	8	192	1956504
16	8	160	1911175
8	8	160	1669896
32	8	128	1603482
16	8	128	1927160
8	8	128	1816467
32	16	64	1939446
16	16	64	1827090
8	16	64	1714734
32	8	64	975422
16	8	64	919958
8	8	64	864494

Table 1 Some Modular Exponentiation Parameters

4.3 HAAP Equation

On a final note, the general timing equation for HAAP is:

$$6(M2) + 6(M1) + 2(B1) + 2(M3) + (B2) + (\text{Table 1})$$

5 CEBus Integration

The CEBus security mechanism is implemented at the application layer, in particularly, within the CAL Element.

This allows authentication regardless of the actual medium used. When a CAL Element received any message, verification of the device requesting the CAL service must be performed via HAAP. The CHALLENGE service allows the CAL Element to send the first portion of an authentication "handshake" to a node requesting a restricted service. See [EIA95] for more details.

5.1 Protocol Flow

To start a SPEKE session, the initiator must send a plaintext CEBus command (via CAL) to a desired service node. Upon packet receipt, the service provider will assemble a Challenge Authentication APDU as described in subsection 5.2. Authentication commences and will stop when either:

1. Any HAAP parameter data is erroneous, or
2. APDU is malformed, or
3. HAAP timer expires, or
4. HAAP succeeds

After a successful authentication, the stacked CEBus command will be executed by the service provider. If the Initiator needs to send additional ciphertext CEBus command, he may do so by encrypting the command with the generated session key. All encrypted messages must use the byte-variable Challenge Authentication APDU format. This encrypted session is valid for only up to 8 seconds before the Challenge Timer runs out. This timer is needed to ensure that minimal ciphertext is transmitted on the CEBus and hence, reducing the possibility of a dictionary attack.

5.2 Format Specification Of HAAP

Figure 4 shows the packet formats used by HAAP. Bytes 1 and 2 are standard BV APDU format as defined in the EIA600 specifications. (See [EIA95] for details.)

MSB: 7	6	5	4	3	2	1	LSB: 0	Byte #
reserved	MODE	ENC	APDU TYPE IDENTIFIER					1
AUTHENTICATION ALGORITHM ID						INVOKE ID		2
SN FLAG	reserved					COUNTDOWN		3
AUTHENTICATION DATA ID			LENGTH IN BYTES					4
AUTHENTICATION DATA 1								5
:								:
AUTHENTICATION DATA m								4+m
AUTHENTICATION DATA ID			LENGTH IN BYTES					5+m
AUTHENTICATION DATA 1								6+m
:								:
AUTHENTICATION DATA n								5+m+n

Figure 4 Home Automation Authentication Protocol APDU

reserved
Always 0
MODE
Always 1 for Byte-Variable (BV) APDU
ENC
Always 1 when using HAAP
APDU TYPE
See [EIA95] for details
ALGORITHM ID
Unassigned until CIC approval
INVOKE ID
A 3 bit incremental identifier used for packet tracking
SN FLAG
Bi-directional Flag to synchronise sequence number
1 for normal Sequence Number authentication,
0 for Sequence Number re-initialisation to destination node
reserved
Always 0
COUNTDOWN
MT Challenge Timer – 3 bit binary down counter
000 for timer lifetime expired – all services are denied
111 for 7 seconds left
AUTHENTICATION DATA ID
HAAP Parameter Identifier
000 CAL package
001 AUTH_HAAP_PRIME
A→B: Gets prime modulus, p
B→A: Sends prime modulus, p
010 AUTH_HAAP_HPRIME
A→B: Sends double hashed prime modulus, HHP
B→A: Sends hashed prime modulus, HP
011 AUTH_HAAP_Q
A→B: Sends public value, Q_A
B→A: Sends public value, Q_B
100 AUTH_HAAP_HHK
A→B: Sends double hashed shared-key, HHK_A
B→A: Not applicable
101 AUTH_HAAP_EHSN
A→B: Sends encrypted double hashed seqnum, $EHHSN_A$
B→A: Sends encrypted hashed seqnum, $EHSN_B$
110 AUTH_HAAP_EPASSWD
A→B: Sends encrypted new password, EP_A
B→A: Sends encrypted new password, EP_B
111 reserved
LENGTH
Length of AUTHENTICATION DATA in binary
AUTHENTICATION DATA
Data field for HAAP parameters

6 Requirements & Limitations

This section highlights some of the HAAP's requirements, known limitations and the rationale behind them.

1. *Need for storage resources on both service requester and provider*
SPEKE optimisations, the sequence number challenge-response authentication and the dynamic N-time password algorithm all need memory storage to store authentication key information. In HAAP, there is no need to transfer volatile memory information onto a permanent storage (e.g. NVRAM) in the event of a power outage. Provisions are also implemented into the protocol to ensure smooth CEBus operations.

2. *Need participation of both parties (i.e. handshaking) to achieve mutual authentication*
   Without participation of both parties, other authentication schemes may be opened to dictionary and/or replay attacks.

3. *Need for physical security for CEBus devices*
   If there are no provisions for physical security, hard coded password S is easily compromised. HAAP dynamic N-time password function could be used secondarily to address this problem since new passwords are randomly chosen by the microcontroller and distributed to other trusted controllers. But users must be aware that a simple power-reset to the device will initialise all passwords to current hardware settings. To explore further, if an adversary can access a device physically, it serves disastrously as a password leakage source since SRAM information can be easily read. Therefore, such risks must not be neglected.

4. *Short modulus simplifies SPEKE discrete log attacks*
   This is one of the goals this paper addresses: to identify the limitations of modern cryptography implemented using low-cost microcontrollers. It must be made known that consumer sensitiveness to product pricing is extremely high. It is uneconomical to fabricate a simple CEBus-compliant lighting device when its controller out costs the material costs of the basic lighting frame. With the rising acceptance of Smart Home Automation systems, demand will force more powerful RISC processor costing down to affordable levels with great prospects.

5. *Legal and policy issues*
   With the exception of IDEA encryption algorithm, HAAP does not use any other intellectual or commercial property that requires royalties. IDEA was chosen in HAAP mainly because it is a very good cipher that has withstood repeated attacks in the academic literature. Unfortunately, Ascom-Systec's has patent licensing terms for using the IDEA algorithm. So at present, alternatives are either DES or RC5, which still provides sufficient protection.

7 Conclusion

The Home Automation Authentication Protocol (HAAP) is specifically designed to provide sufficient authentication facilities to be used for home automation (HA) devices. The protocol accomplishes its strength from SPEKE strong password method, sequence number challenge-response authentication, dynamic N-time password and on-line clogging detection and avoidance. SPEKE generator values are extensively reviewed to provide realistic figures when implemented using low-cost micro-controllers. Integration procedure of HAAP into the CEBus protocol stack is also shown to demonstrate the feasibility of using HAAP with practical HA standard.

References

[BEN94] Stuart Bennett, "Real-Time Computer Control - An introduction", 2nd Edition, 1992

[EIA95] EIA/IS-60, "CEBus Standard", Jun 92 and revised Feb 1995

[Itln94] Intellon, "CENode Network Interface Board Specification v1.1", 1994

[Jab96] David P. Jablon, "Strong Password-Only Authenticated Key Exchange", Integrity Sciences, Inc, Sep 1996

[Jab97] David P. Jablon, "Extended Password Key Exchange Protocols Immune to Dictionary Attack", Integrity Sciences, Inc, 1997

[LSS98] Leong See Sum, "Home Automation Authentication Protocol", http://www.ntu.edu.sg/home/aschvun/haap, 1998

[LMM92] X. Lai, J.L. Massey, S. Murphy., "Markov ciphers and differential cryptanalysis. In Advances in Cryptology — Eurocrypt '91, pages 17–38", Springer-Verlag, 1992

[Riv92] R.L. Rivest, "RFC 1321:The MD5 Message-Digest Algorithm", Internet Activities Board, Apr 1992

[RSA96] Paul Fahn et al., "Frequently Asked Questions About Today's Cryptography", RSA Laboratories, 1996

[SchBr1] Bruce Schneier, "Applied Cryptography", ISBN 0-471-59756-2

[SRP97] Thomas Wu, "The Secure Remote Password Protocol", http://srp.stanford.edu/srp, 1997

[Vun97] Nicholas Vun et al, "Information Security of Home Automation Systems", Southeast Asia Fire & Security, page 29, Nov/Dec 1997

Biography

Leong See Sum received his B.A.Sc (Hons) degree in Computer Engineering from Nanyang Technological University, Singapore in 1998. Since 1996, he has been working on real-time embedded systems. His interests include high-speed digital/analog systems, networking and computer security. He is now working as an Information Systems Officer in Systems & Computer Organisation, MINDEF.

Nicholas C. H. Vun completed his B.Eng (Hons) and M.Eng.Sci degree in Electrical and Computer System Engineering in Monash University, Australia in 1984 and 1987 respectively. He joined Nanyang Technological University in 1991 and is currently a Senior Lecture in its School of Applied Science faculty. His research interests include real time embedded system design and applications, home automation, power electronics and information security.