Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks

Dan Harkins
Aruba Networks
Sunnyvale, California, 94089, USA Email: dharkins@arubanetworks.com

Simultaneous Authentication of Equals (SAE)

***Abstract*—We propose a simple protocol for authentication using only a password. The result of the protocol is a cryptographically strong shared secret for securing other data– e.g. network communication. SAE is resistant to passive attack, active attack, and dictionary attack. It provides a secure alternative to using certificates or when a centralized authority is not available. It is a peer-to-peer protocol, has no asymmetry, and supports simultaneous initiation. It is therefore well-suited for use in mesh networks. It supports the ability to tradeoff speed for strength of the resulting shared key. SAE has been implemented for 802.11-based mesh networks and can easily be adapted to other wireless mesh technology.**

I. Introduction

The motivation for this work is to address the problem where there exists two entities that share a password[1] and wish to authenticate each other and derive a strong, secret and shared key. There also exists an active and powerful adversary whose capabilities include: evesdropping on conversations between the two entities; inserting itself into the conversation between the two entities such that all messages sent between the two must go through it; replaying, dropping, and modifying messages sent between the entities; and, enumerating, off-line, through entries in a dictionary[2] from which the password is drawn.

This problem, password authenticated key exchange, comes from papers by Bellovin and Merrit ([1], [2]) which proposed the first solutions. Subsequently, many other protocols have been presented that solve also this problem, including [3], [4], [7], [9], [10], [11], [14], [15].

Another solution is described here. In addition to being another solution to the password authenticated key exchange problem, this solution also uses symmetric messaging and is well-suited for peer-to-peer applications like mesh networking where the entities involved have no fixed roles that govern their behavior– in other words, there is no $initiator$ or $responder$, and there is no $client$ or $server$. In SAE either party can initiate the conversation or both parties may initiate simultaneously.

[1]The use of the word $password$ in this paper does not imply any particular representation and is just a euphemism for a shared secret in any form.

[2]There is no assumption that a $dictionary$ in this case is limited to only contain words, it is generally just a finite pool of secrets.

II. Requirements for Password-Authentication in a Mesh Network

A. Resistance to Attack

A protocol that employs a password for entity authentication must meet a basic set of security requirements. These include:

- An attacker must not be able to glean any information about the password or the resulting shared secret from a passive (evesdropping) attack.
- The only information an attacker can obtain from a single active attack is whether a single guess of the password is correct or not.
- Compromise of the shared secret from a previous run of the protocol won't help an attacker in an attack on another run of the protocol (the "Denning-Sacco attack").
- Compromise of the password will not allow an attacker to know anything about the shared secret from a previous run of the protocol ("forward secrecy").

B. Peer-to-Peer Not Lock-step

Generally protocols start with one side being prompted to send an initial message. Upon receipt of that message the other party responds. Messages are send back and forth in this fashion, each side waiting for the other party's next message before sending its own next message, until the protocol terminates. That is an example of a lock-step protocol. One entity initiates a conversation– e.g. a client– and the other entity– e.g a server– remains in a quiescent state until it is initiated to at which time it responds.

In [5], Bellare and Rogaway formalize protocol conversation as a time-based sequence of messages sent by an oracle Π:

$$\Pi : (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), ..., (\tau_m, \alpha_m, \beta_m)$$

where at time τ_i the oracle is asked α_i and responds β_i.

A $matching\ conversation$ of a protocol is then one where every message sent by an oracle, Π_A, except possibly the last, is delivered to another oracle, Π_B, with the response to that message being returned to Π_A as its own next message.

Typically a protocol starts with one oracle being given a NULL input at time τ_0 and that oracle is called an "initiator" with the other oracle called a "responder".

Such a protocol is, though, unsuitable for mesh networks since there are no "initiators" or "responders" or "clients" or "servers" in a mesh. While it is possible to force mesh nodes into strict roles, that is unnatural and the result is a fragile mesh

978-0-7695-3330-8/08 $25.00 © 2008 IEEE
DOI 10.1109/SENSORCOMM.2008.131

that forms slowly and has difficulty reforming after unexpected partition.

In a mesh network the node initiating the protocol is the one that discovered it's neighbor first and it is possible for mesh nodes to discover each other simultaneously and have their initial messages cross in the ether. To build a robust mesh, protocols necessary for forming the mesh, including key exchange and authentication protocols, must support simultaneous initiation, they must be peer-to-peer and not lock-step.

For a protocoll to be peer-to-peer, a conversation must remain a "matching conversation" regardless of which oracle initiated and even when both oracles initiated simultaneously for a particular conversation. The conversation α_1,β_1, α_2,β_2 over time $\tau_0 < \tau_1 < \tau_2 < \tau_3 < \tau_4$ can be represented as:

$$\Pi_A : (\tau_0, NULL, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, NULL)$$
$$\Pi_B : \quad (\tau_1, \alpha_1, \beta_1), \ (\tau_3, \alpha_2, \beta_2)$$

or equivalently as:

$$\Pi_A : \quad (\tau_1, \beta_1, \alpha_1), \ (\tau_3, \beta_2, \alpha_2)$$
$$\Pi_B : (\tau_0, NULL, \beta_1), (\tau_2, \alpha_1, \beta_2), (\tau_4, \alpha_2, NULL)$$

or, also equivalently, as:

$$\Pi_A : (\tau_0, NULL, \alpha_1), (\tau_1, \beta_1, \alpha_2), (\tau_2, \beta_2, NULL)$$
$$\Pi_B : (\tau_0, NULL, \beta_1), (\tau_1, \alpha_1, \beta_2), (\tau_2, \alpha_2, NULL)$$

If all three representations of the conversation are equivalent and all result in identical shared state then the protocol in which the conversation took part is a peer-to-peer protocol.

An oracle in an authenticated key exchange protocol $accepts$ if and only if it has authenticated its peer and generated a shared secret value. It $rejects$ if it has failed to $accept$. It should be mentioned that accepting is different than terminating the protocol. When an oracle terminates it has received all information it expects and will send no other messages [4]. The notion of accepting at one particular time during the conversation and terminating later is important for peer-to-peer protocols because an oracle can be in the situation of having sent all messages but not having received all messages, and vice versa, depending on the order of the messages in the particular flow.

III. The SAE Protocol

As in many other descriptions of key exchange protocols the parties involved in SAE are $Alice$ and Bob. These names are used as placeholders for their unique identities, which could be, for example, their MAC addresses. The result of these entities performing the protocol is an authenticated and shared secret.

The adversary is called Eve, and her goal is to attack the protocol. Eve will be successful if she can learn the password, if she can learn the resulting shared key, or if she can make Alice or Bob erroneously believe her or his conversation has successfully completed.

An alternative way of thinking about the security of SAE is that the only way Eve can get Alice and Bob to $accept$ is by faithfully relaying messages between them.

A. Finite Cyclic Groups

SAE requires a finite cyclic group in which the "Discrete Logarithm problem" is known to be computationally intractable. Such groups can be based on exponentiation of integers modulo a prime (prime modulus groups) or based on elliptic curves over a finite field (elliptic curve groups).

The strength of the shared key resulting from a successful run of SAE depends on the specific group being used. The larger the work factor to solve the discrete logarithm problem in that group the stronger the key. Of course, the larger the prime (for prime modulus groups) or larger the group (for elliptic curve groups) the more computational resources are necessary to complete a run of SAE. It is therefore possible to trade speed of execution for strength of key when choosing a group.

There are techniques to solve the discrete logarithm problem in prime modulus groups that are more efficient than the best known techniques for elliptic curve groups. Therefore, for a given key strength it is possible to use elliptic curve groups of a smaller size than prime modulus groups.

Both types of groups have two group operations defined, a scalar operation and an element operation. For elliptic curve groups the scalar operation is multiplication of a point on the curve by a scalar, and the element operation is addition of two points on the curve. For prime modulus groups the scalar operation is exponentiation of a generator to some scalar power, and the element operation is the modular multiplication of two elements.

SAE also requires an operation to take the inverse of a group element. For elliptic curve groups, a point on the curve is the inverse of a different point on the curve if their sum is the "point at infinity". For prime modulus groups, an element is the inverse of a different element if their product modulo the group prime is one (1).

B. Notation

A finite cyclic group is denoted Γ. A generator of Γ is G and its order is r. Elements in Γ are written in upper-case while numbers are written in lower-case. The scalar group operation is denoted with $\bullet$ and the element group operation is denoted with $\diamond$. For example:

$$A = (a \bullet G) \diamond Q$$

Concatenation is symbolized by $|$.

Alice and Bob generate identical information. Alice's information will be denoted by a A subscript, $info_A$, while Bob's information will be denoted by a B subscript, $info_B$.

C. Assumptions

It is assumed that Alice, Bob, and Eve have a shared knowledge of the following:

- an ordering function, L, that returns the "greater" of two identities. Since identities cannot be identical L cannot return "equal".
- a random function with a blocksize of s bits:

$$H : \{0,1\}^* \longrightarrow \{0,1\}^s$$

- a key derivation function that takes an arbitrary string and stretches it to a given length:

$$bigstr = KDF(smallstr, length)$$

- a finite cyclic group, Γ.
- a bijective function F which takes an element in the group and returns a number.

In addition, Alice and Bob share a $password$ that is unknown to Eve.

The security of SAE depends on the finite cyclic group used and on the following assumptions about function H:

1) H is a "random oracle". The output of H is indistinguishable from random by an adversary.
2) H is a one-way function. Given y = H(x) it is infeasible to determine x.
3) for some given input to H each of the 2^s possible outputs are equally probable.

D. Protocol Exchange

The protocol begins upon discovery of a peer. Prior to sending any messages Alice and Bob select an element in the group, Γ, based on the password and their identities. The specific technique in fixing the password element, PWE, depends on the type of finite cyclic group used.

1) Elliptic Curve Group: A hunt-and-peck technique is used to deterministically select a "random" point on the curve.

$i = 0$
repeat
  $i = i + 1$
  if $L(Alice, Bob) = Alice$ **then**
    $pwdseed = H(Alice \mid Bob \mid password \mid i)$
  else
    $pwdseed = H(Bob \mid Alice \mid password \mid i)$
  end if
  $x = (KDF(pwdseed,\ len)) \bmod p$
  solve for y using the equation for the curve with x
  if $pwdseed$ is odd **then**
    $PWE = (x, -y)$
  else
    $PWE = (x, y)$
  end if
until PWE is on the curve

where p is the prime of the curve and len is the length of p.

2) Prime Modulus Group: For prime modulus groups PWE is fixed directly through exponentiation.

if $L(Alice, Bob) = Alice$ **then**
  $pwdseed = H(Alice \mid Bob \mid password)$
else
  $pwdseed = H(Bob \mid Alice \mid password)$
end if
$pwdvalue = (KDF(pwdseed,\ len)) \bmod p$
$PWE = pwdvalue^{((p-1)/r)} \bmod p$

where p is the group prime, r is the order, and len is the length of p.

After the password element is fixed, each party chooses two random numbers, $rand$ and $mask$. An element, $elem$, and a scalar, $scal$, are then produced:

$scal = (rand + mask) \bmod r$
$elem = inverse(mask \bullet PWE)$

where $inverse()$ is the inverse function for the finite cyclic group, an r is the group order.

Each party sends its element and scalar to the other:

Alice $\rightarrow$ Bob: $scal_A, elem_A$
Alice $\leftarrow$ Bob: $scal_B, elem_B$

A shared secret element is computed using one's $rand$ and the other peer's element and scalar:

Alice: $K = rand_A \bullet (scal_B \bullet PWE \diamond elem_B)$
Bob: $K = rand_B \bullet (scal_A \bullet PWE \diamond elem_A)$

Both parties compute an intermediate shared key, $k = F(K)$, and an authentication confirmation token, tok, which is sent to the other party, Alice sends tok_A to Bob and Bob sends tok_B to Alice:

$$tok_A = H(k \mid\ F(elem_A) \mid scal_A \mid F(elem_B) \mid scal_B)$$
$$tok_B = H(k \mid\ F(elem_B) \mid scal_B \mid F(elem_A) \mid scal_A)$$

Both parties verify the value of the other party's authentication confirmation token and, if valid, create the shared key:

$$secret = H(k \mid F(elem_A \diamond elem_B) \mid (scal_A + scal_B) \bmod r)$$

IV. SAE: A Peer-to-Peer Protocol

The messages sent by Alice and Bob are identical and, to the fullest extent possible, do not depend on each other. It is not necessary for Bob to wait for Alice's first message before he sends his (and vice versa), nor is it necessary for Alice to wait for Bob to send his second message before she sends hers (and vice versa).

If the first message in SAE sent by either peer is called a "commit" and the second a "confirm", the following protocol rules can be described:

- either peer can $commit$ at any time
- a peer can $confirm$ only after it has $committed$ and the other peer has $committed$
- a peer can $accept$ after its peer has $confirmed$ and the confirmation has been verified
- a peer can $terminate$ after it has $confirmed$ and $accepted$

With these rules in mind it is easy to see that an instance of a traditional lock-step "matching conversation" of this protocol with Alice as the "initiator"is identical to a lock-step instance where Bob is the "initiator". It is also identical to an exchange where Alice's first message and Bob's first message are exchanged simultaneously (they both view themselves as the "initiator") and Alice's second message and Bob's second mesage are exchanged simultaneously. Provided the aforementioned rules are adhered to, the order of the messages

forming Alice's and Bob's matching conversations can be altered and the resulting shared secret will still be identical.

Clearly, SAE is a peer-to-peer protocol.

V. Security of SAE

This is not a formal proof; it is a demonstration of SAE's resistance to attack. A formal proof of security in the random oracle model is the subject of on-going work by the author.

SAE uses the symmetric trust model. Both parties to the exchange must have a common view of the password. It deserves mentioning that the view of the password could, in fact, be derived from a one-way function of a raw password. Employing such a technique would add an additional burden to an attacker who has stolen a mesh point provisioned with a common view of the password.

A. Resistance to Attacks

To demonstrate the security of SAE it must be shown that: 1) the protocol works and is not trivially secure, that when run honestly between two parties the result is a shared secret; 2) passive attacks are computationally infeasible; 3) active attacks are infeasible; 4) dictionary attacks are not possible; 5) compromise of the password does not compromise older sessions ("forward secrecy"); and, 6) compromise of short-term secret session keys does not compromise other sessions (the "Denning-Sacco attack").

1) Not Trivially Secure: The protocol results in a shared secret when run between two honest peers (without Eve attacking the exchange). This can be seen by inspection: Alice computes

$$\begin{aligned} K &= rand_A \bullet (scal_B \bullet PWE \diamond elem_B) \\ &= rand_A \bullet rand_B \bullet PWE \end{aligned}$$

and Bob computes

$$\begin{aligned} K &= rand_B \bullet (scal_A \bullet PWE \diamond elem_A) \\ &= rand_B \bullet rand_A \bullet PWE \end{aligned}$$

2) Passive Attack: A passive attacker (or "benign attacker") merely relays messages from one honest player to another. The attacker is able to inspect and store each message but must faithfully relay them from Alice to Bob and back again. Success in this attack is achieved by gleaning information from the messages and obtaining an advantage to determine the shared key or to determine the password.

a By adding $mask$ to $rand$ modulus the order of the group the secret random values of Alice and Bob are effectively hidden from a passive attacker. Eve can see $scal_A$ ($scal_B$) but there are approximately r different ways that two random numbers can be summed modulus r. Eve's success at guessing the combination is essentially the same as guessing either random number. Therefore we say her ability to determine either $rand_A$ or $rand_B$ is infeasible.

b Eve can see both sets of elements and scalars. She can guess passwords to compute PWE and attempt to compute: $(scal_A \bullet PWE \diamond elem_A)$ or $(scal_B \bullet PWE \diamond elem_B)$ but since she does not know either $rand_A$ or $rand_B$ she is unable to use tok_A or tok_B to verify whether a guessed password was correct.

A passive attack cannot succeed in determining the password or the shared key.

3) Active Attack: An active attacker is one that modifies or forges messages sent to an honest player. Eve can attack either Alice or Bob but the peer-to-peer nature of the protocol makes the selection irrelevant. Let's assume Alice is the honest peer being attacked by Eve.

a Eve can launch her attack against Alice after an honest Bob has sent $scal_B, elem_B$.
- This would require Eve to construct a tok_B that Alice would successfully verify. Eve cannot know the random secret numbers, $rand_B$ and $mask_B$, that the honest Bob used in constuction of $elem_B$ and $scal_B$. Eve similarly cannot know Alice's random secret value, $rand_A$, so this attack is reduced to that of a passive attacker.
- She can try to construct a random tok_B in the hope that Alice accepts it but due to the assumptions about function H this is infeasible.

b Eve can launch her attack by generating $scal_B$ and $elem_B$ herself. After Alice sends $scal_A$ and $elem_A$ Eve sends the bogus $scal_B$ and $elem_B$ and waits for tok_A. There are three possible goals here: 1) to learn the shared key; 2) to determine the password; or, 3) to generate a tok_B value that would cause Alice to accept;

1) For Eve to learn the shared key she must be able to compute K. She knows $rand_B$ since she created it, and she was given $elem_A$ and $scal_A$. She can run through guesses in the dictionary to compute K and hopefully use the resulting k to verify tok_A. But Eve committed to a password guess when generating her bogus $elem_B$ and Alice used that committed guess to compute her version of k, which was used to construct tok_A. So even if Eve guessed the right password on the first try when computing

$$K = rand_B \bullet (scal_A \bullet PWE \diamond elem_A)$$

her resulting key will be different than Alice's and she would, therefore, have no way of knowing whether her guess is right or not.

Eve could use the password she committed to in the construction of her bogus $elem_B$ and then run through passwords, to generate PWE, while trying to compute Alice's key but that would require knowledge of $rand_A$ which we have already shown is not possible.

Eve cannot use an active attack to determine the shared key.

2) PWE is an element in the finite cyclic group. There exists a scalar, q, such that $PWE = q \bullet G$, where G is the generator for the group. Knowledge of q can

enable an off-line dictionary attack to determine the password.

To do this Eve generates random numbers $rand_B$ and $mask_B$ and computes

$$scal_B = (rand_B + mask_B) \bmod r$$
$$elem_B = mask_B \bullet G$$

(Note that Alice has no way of telling that $elem_B$ was computed incorrectly). After Alice sends $scal_A$ and $elem_A$ Eve sends the bogus $scal_B$ and $elem_B$. Alice will compute

$$K = rand_A \bullet (scal_B \bullet PWE \diamond elem_B)$$

and generate tok_A. Because $elem_B$ was based on G and not PWE, Eve now knows that

$$K = (rand_B * q + mask_B * q + mask_B) \bullet (rand_A \bullet G)$$

and that

$$rand_A \bullet G = scal_A \bullet G - (1/q \bmod r) \bullet elem_A$$

where r is the order of the group. Eve now has everything she needs: she knows $rand_B$ and $mask_B$ and she knows $scal_A$ and $elem_A$. She can run through the dictionary making guesses of the password, generating the element PWE, determine q from PWE, generate a candidate $rand_A \bullet G$ and then a candidate K and see if she is able to verify tok_A. When she is successful she knows the password.

This attack is predicated on knowledge of q. But to determine q from PWE it is necessary for Eve to perform a discrete logarithm which we have assumed in infeasible. Eve cannot use an active attack to learn the password.

- For Eve to generate a tok_B value that Alice would accept she would need to be able to know k. We have already established that is not possible. Eve could guess at a tok_B value and hope Alice accepts but due to the assumptions on function H that is infeasible.

It is not possible to use an active attack to learn the shared key or to cause one party to erroneously believe the exchange finished successfully. The probability of using an active attack to learn the password is $1/D$ where D is the size of the dictionary.

4) Dictionary Attack: For Eve to discover the password she needs to have some way of determining that her guess was correct or not. Since she has no way of running through the dictionary after a single attack she is reduced to making a guess, performing the protocol, verifying the guess and, if unsuccessful, trying another guess with another run of the protocol.

Eve's advantage grows primarily through interaction with Alice and not through computaion. This is a demonstration of resistance to dictionary attack [4].

5) Forward Secrecy: Compromise of the long-term shared secret, the password, does not provide an advantage to an attacker in determining the shared secret from earlier runs of SAE. This is because the shared secret is also based on random contributions by each party that are unknown to an attacker. SAE therefore provides "forward secrecy".

It should be noted, though, that compromise of the password can enable attack against all future runs of the protocol because a man-in-the middle attack cannot be detected by either side.

6) Denning-Sacco Attack: The shared secret resulting from SAE is output from the random function, H. Due to the assumptions made about H it is infeasible to determine the inputs to it, including the intermediate shared key, from its output.

The intermediate shared key is based on the password as well as the random values contributed by each of the accepting peers. Compromise of the intermediate shared key would not provide an advantage to an attacker in determining a different shared secret (or different intermediate shared key) from another run of the protocol. This is due to the random nature of the contributions each side makes to the exchange.

SAE is therefore resistant to the "Denning-Sacco attack".

Compromise of one of the secret random values contributed by each participant in the exchange, though, can enable a dictionary attack to recover the password. To do this it would be necessary to record the exchange from which a secret random value was obtained. It would then be possible to enumerate potential passwords from the dictionary, generate PWE, and compute the intermediate shared key from the compromised random value, PWE, and the scalar and element provided by the other party. When it is possible to verify the authentication confirmation token from the other party the guessed password is correct. Forward secrecy, though, is still provided.

B. Password Strength

The unpredictability of the password and the size of the dictionary from which it was drawn determine the strength of the password. The stronger the password, the more unlikely it is that it will be guessed. Since SAE is resistant to dictionary attack the stronger the password the more active attacks an adversary must launch in an effort to learn it. Regardless of the strength of the password, though, the only way for an attacker to learn the password is through repeated active attack. This has significant benefits to mesh networks.

Passwords have been used for decades and people are comfortable using them. That, combined with a reluctance among many network administrators to deploy a certification authority, means that passwords will be deployed to secure mesh networks. Deploying pair-wise passwords in a mesh of n nodes is an $O(n)$ operation while deploying a single, shared, password on all mesh nodes is an $O(1)$ operation and many operators will choose the latter. Since the password will need to be entered repeatedly with a low probability of error the password will, most likely, be something easy for the

administrator to remember and enter. In other words, it will most likely be cryptographically weak ([12]).

The combination of weak and shared can be tragic for a password but the dictionary resistance of SAE mitigates that to some extent.

The fact that repeated active attack is the only way to learn the password means that the attacker is not able to increase her advantage by distributing her attacks against many mesh nodes instead of repeatedly attacking a single mesh. Therefore, sharing the password among mesh points does not significantly alter the security of the mesh.

Obviously, password strength matters and the weaker the password the fewer the attacks needed for the adversary to gain a realistic advantage. But countermeasures can be employed to possibly refuse new authentication attempts after a certain number of failures and failed attempts can be logged to enable an operator to detect an attack against the mesh and respond appropriately.

SAE is a robust, secure authentication protocol for mesh networks even in the presence of weak and shared passwords.

VI. SAE in 802.11-based Mesh Networks

SAE is defined generally and is applicable to all types of mesh networks. It has been adopted by Task Group "s" (mesh networking) of the IEEE 802.11 Working Group. To use SAE it is necessary to define the framing of the components each side sends, how function H is constructed, and how the finite cyclic group is agreed upon.

A. Framing SAE

SAE is performed using 802.11 authentication frames. A "commit" message has an authentication sequence number of one (1) and a "confirm" message has an authentication sequence number of two (2). The contents of each message are encoded into the frame after the authentication header. For a commit message the scalar is first, followed by the element.

If an elliptic curve group is used the element is a point on the curve and is encoded as the x-coordinate followed by the y-coordinate.

B. Definition of the Random Function

The random function H is assumed to be a "random oracle" as described in [6]. That paper describes the properties a random oracle must have and suggests several ways to instantiate a random oracle using a strong hash function.

The particular instantiation for 802.11 is repetition of input to SHA-256 ([13]): $H(x) = SHA256(x \mid x)$.

C. Negotiation of the Group

The Internet Assigned Numbers Authority (IANA) manages a repository of finite cyclic groups for the Internet Key Exchange [8] and these groups satisfy the requirements of SAE. SAE in 802.11 takes advantage of this repository by encoding the group into the authentication algorithm field of an 802.11 authentication frame. If the high-order bit of the authentication algorithm field is set it indicates SAE authentication and remaining bits are used to identify a group in the IANA registry.

VII. Conclusion

A secure, peer-to-peer, and password-based key exchange has been presented. This protocol is well-suited for mesh networks and can support a wide variety of security requirements. It is secure against passive attack, active attack, and dictionary attack.

The protocol is presented in a general fashion and can easily be adapted to any communications media on which mesh networks are formed. An example of SAE in an 802.11-based mesh network is demonstrated.

Acknowledgment

The author would like to thank Hideyuki Suzuki and Scott Fluhrer for their insight and analysis that identified flaws in earlier versions of SAE. He would also like to thank David McGrew for helpful suggestions during numerous conversations on the problem of password-authenticated key exchange. Thanks to Merwyn Andrade for encouraging this work.

References

[1] S. Bellovin and M. Merritt, Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks. Proc of the Symposium on Security and Privacy, pages 72-84, IEEE, 1992

[2] S. Bellovin and M. Merrit, Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise, Proceedings of the 1st Annual Conference on Computer and Communications Security, ACM, 1993

[3] V. Boyko, P. MacKenzie, S. Patel, Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman, Advances in Cryptology– Eurocrypt 2000, Springer-Verlag, 2000

[4] M. Bellare, D. Pointcheval and P. Rogaway, Authenticated Key Exchange Secure Aga inst Dictionary Attacks, Advances in Cryptology– Eurocrypt 2000, LNCS vol 1807, Springer-Verlag, pp. 139-155, 2000

[5] M. Bellare, P. Rogaway, Entity Authentication and Key Exchange, Advances in Cryptology– Crypto '93 Proceedings, Springer-Verlag, August 1993

[6] M. Bellare, P. Rogaway, Random Oracles are Practicle: A Paradigm for Designing Efficient Protocols, Proceedings of First ACM Conference on Computer and Communications Security, November 1993

[7] L. Gong, T.M.A. Lomas, R.M. Needham, J.H. Saltzer, "Protecting Poorly Chosen Secrets from Guessing Attacks, IEEE Journal on Selected Areas In Communications, Vol.11, No.5, June 1993

[8] D. Harkins, D. Carrel, The Internet Key Exchange, Request For Comments (RFC) 2409, November 1998

[9] D. Jablon, Strong Password-Only Authenticated Key Exchange, ACM Computer Communi cations Review, October 1996

[10] J. Katz, R. Ostrovsky, M Yung, Efficient Password-Authenticated Key Exchange using Human-Memorable Passwords, Advances in Cryptology– Eurocrypt 2001, Springer-Verlag, 2001

[11] S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. Proc. of the Security Protocols Workshop, LNCS 1361, Springer-Verl ag, 1997

[12] R. Morris, K. Thompson, Password Security: A Case Study, Communications of the ACM, Vol.22, No.11, November, 1993

[13] National Institute of Standards and Technology, Secure Hash Standard, Federal Information Processing Standard (FIPS) 180-2

[14] S. Vaudenay, Secure Communications over Insecure Channels Based on Short Authenticated Strings, Advances in Cryptology– Crypto 2005, Springer-Verlag, 2005

[15] T. Wu, The Secure Remote Password Protocol, Proceedings of the Internet Society Symposium on Network and Distributed System Security, pages 97-111, 1998