

# Ultimate Solution to Authentication via Memorable Password

Taekyoung Kwon\*

Updated on May 30, 2000

- *Contribution to the IEEE P1363 study group for Future PKC Standards* -

## Abstract

A new password authentication and key agreement protocol called AMP is proposed in a provable manner. Human-memorable password authentication is not easy to provide over insecure networks due to the low entropy of the password. A cryptographic protocol, based on the public-key cryptography, is the most promising solution to this problem. AMP provides the password-verifier based authentication and the Diffie-Hellman key agreement, securely and efficiently. AMP is easy to generalize in any other cyclic groups. Verifier-based protocols allow the asymmetric model in which a client possesses a password, while a server stores its verifier. AMP is actually the most efficient protocol among those protocols. Several variants of AMP are also proposed. They are called  $AMP^i$ ,  $AMP^n$ ,  $AMP^+$ ,  $AMP^{++}$ . Among them,  $AMP^n$  is a pure password-based protocol rather than a verifier-based protocol. In the end, we give a rigorous comparison to the related protocols.

The protocols described in this contribution are from the papers, *Ultimate Solution to Authentication via Memorable Password* [23].

---

\*741 Soda Hall, Computer Science Division, EECS, University of California, Berkeley, CA 94720, tkwon@cs.berkeley.edu or ktk@emerald.yonsei.ac.kr. He is supposed to join datawave.co.kr since September, 2000.

# 1 Introduction

Entity authentication is one of the most important security functions. It is necessary for verifying the identities of the communicating parties when they initiate a connection. This function is usually provided in combination with a key establishment scheme such as key transport or key agreement between the parties. For user authentication, three kinds of approaches exist which may be combined for sophisticated use; knowledge-based authentication, token-based authentication, and biometric authentication. Among them, the knowledge-based scheme needs knowledge-proofs so that it is only for human mind ( $\approx$  memory). Actually, it is the most widely-used method due to the advantages of simplicity, convenience, adaptability, mobility, and less hardware requirement. It requires users only to remember and type in their knowledge such as a password or PIN(personal identification number). Therefore, users are allowed to move conveniently without carrying hardware tokens; the user knowledge is also useful for unlocking such hardware tokens. However, a complex problem with this password-only authentication is that a human-memorable password having low entropy allows malicious guessing attacks. The problem becomes much more critical in an open distributed environment. A cryptographic protocol is the most promising solution to it.

PASSWORD PROTOCOLS.<sup>1</sup> Since the first scheme, LGSN[26], was introduced in 1989, many protocols have followed it. Among them, EKE[7] was a landmark of certificate-free protocols though it gave several strong constraints due to the symmetric encryption of a public-key. One variant of EKE, DH-EKE[7], introduced the password authentication and key agreement, and was “enhanced” to A-EKE[8] that was the first verifier-based protocol to resist a password-file compromise and to accommodate salt. Note that there is an earlier work which describes the use of salt[39] but we consider the EKE families as roots. GLNS[15] is enhanced from LGSN. Due to the inefficiency and constraints of older schemes, several modifications and advanced descendents were spawned[38, 1, 37, 16, 21, 18, 19, 35, 40, 17]. However, some of them have been broken and some are still being cryptanalyzed[2, 13, 32, 9]; most were inadequate for the security proof due to ad-hoc methods of protecting the password. In the mean time, OKE was introduced as the first provable approach based on the work of Bellare and Rogaway[27, 3] and was followed by SNAPI[28]. Halevi and Krawczyk’s work also intended a provable approach but their protocol requires users to keep some additional information called a public password[17]. Most recently, AuthA and PAK have been introduced separately[5, 6, 10] and they show the provable approach in this area is getting matured[4].

A family of the password-verifier based protocol is composed of A-EKE, B-SPEKE, SRP, GXY, SNAPI-X, AuthA, and PAK-X[8, 19, 40, 22, 28, 6, 10]. The verifier-based protocols allow the asymmetric model in which a client possesses a password, while a server stores

---

<sup>1</sup>Readers are referred to Figure 7 attached in Appendix of this document. Jablon’s work[18] is recommended as the best tutorial for the password protocol study while Wu’s work[40] is the best for the verifier protocol study. Bellare and Rogaway’s work[3] is the fundamental of the provable approach.

its verifier rather than the password itself. A-EKE was the first verifier-based version augmented from DH-EKE[8]. B-SPEKE was augmented from SPEKE which was efficient without a verifier[18, 19]. SRP was efficient and practical even with a verifier[40]. GXY was derived from SRP for agreeing on the Diffie-Hellman exponential[22]. SNAPI-X was augmented from SNAPI and it was fully provable in the random oracle model[28]. AuthA was newly proposed but very similar to the previous protocols[6]. PAK-X was enhanced from PAK[10] and it was so recent version of those protocols that we could find it when we were on finishing this paper. We will give a rigorous comparison of our protocol, AMP, to those protocols.

CONTRIBUTION. Our goal is to design the password-verifier based authentication protocol in a provable manner, which allows a server-compromise resistance via verifier and a secure key agreement based on the Diffie-Hellman scheme. The protocol must be efficient and easy to generalize by utilizing the main group operation. In addition, we give a rigorous comparison of all verifier-based protocols. Our protocol was actually called u-AMP(Ultimate Authentication and key agreement via Memorable Password) because it is an ultimate result of AMP<sup>2</sup> research project[23]<sup>3</sup>. However, we call it simply AMP. Regarding several functional issues, we also present four major variants of AMP. They are called AMP<sup>i</sup>, AMP<sup>n</sup>, AMP<sup>+</sup>, AMP<sup>++</sup>. Several minor variants also can be considered but they are explained implicitly. Actually, AMP is the most efficient protocols among the existing verifier-based protocols.

## 2 AMP Protocol Design

### 2.1 Preliminaries

Since AMP is typically the two party case, we use *Alice* and *Bob* for describing a client and a server, respectively. *Eve* indicates an adversary whether she is passive or active.  $\pi$  and  $\tau$  denotes a password and salt, respectively.  $\doteq$  means a comparison of two terms, for example,  $\alpha \doteq \beta$ . Let  $\{0, 1\}^*$  denote the set of finite binary strings and  $\{0, 1\}^\infty$  the set of infinite ones.  $\lambda$  implies the empty string.  $k$  is our security parameter long enough to prevent brute-forcing while  $l(k) \geq 2k$ ,  $\omega(k) \leq \frac{2}{3}k$ , and  $t(k) \leq \frac{1}{3}k$ .  $h() : \{0, 1\}^* \rightarrow \{0, 1\}^{l(k)}$  means a collision-free one-way hash function.  $\varphi() : \{0, 1\}^* \rightarrow \{0, 1\}^{\leq 2l(k)}$  means a one-way function. All hash functions are assumed to behave like random oracles[3]. Note that we abbreviate a modular notation,  $\text{mod } p$ , for convenience hereafter.

THE VERIFIER. A verifier  $\mathcal{V}$  is the information computed from salt  $\tau$  and password  $\pi$ . It is composed of an explicit verifier  $\nu$  retained by a server and an implicit verifier  $v$  obtained by a client.  $\nu$  is computable from  $v$  whereas the reverse is infeasible in polynomial time.

---

<sup>2</sup>AMP implies a password amplifier which means a method of strengthening the low-entropy password.

<sup>3</sup>AMP is the postdoctoral research of the author at University of California, Berkeley.

- explicit verifier :  $\nu = g^v$ ,
- implicit verifier :  $v = \varphi(\tau, \pi)$ .

If *Eve* knows  $\tau$  and  $\nu$ , she can find  $v$  by the guessing attack or impersonates a server; this is inevitable in all verifier-based protocols. Two verifiers are used for preventing the server compromise attack in the way that a compromised password-file shows the explicit verifier whereas the implicit verifier is actually necessary for authentication.  $\varphi()$  is a one-way function that merely perturbs and expands  $\tau$  and  $\pi$  into a secure exponent[31],  $\varphi() \in \{0, 1\}^{\leq 2l(k)}$ .

SALT. We assume salt  $\tau$  is disclosed in every protocol run. For example,  $\tau$  is retained by *Bob* but transmitted to *Alice* in every protocol run. This is a typical case in the existing salt system such as the Unix password file. We call this explicit salt. However, we can also allow implicit salt by making both parties get  $\tau$  respectively without exchange, e.g., from their identities[6, 24], though it is not favorable for upgrading the existing salt systems.

RANDOM ORACLE. We assume random oracles  $h_i() : \{0, 1\}^* \rightarrow \{0, 1\}^{l(k)}$  for  $i \in [1, 5]$ . If *Eve* sends queries  $Q_1, Q_2, Q_3, \dots$  to the random oracle  $h_i$ , she can receive answers  $h_i(Q_j)$ , all independently random values, from the oracle. Note that  $\varphi()$  is in  $\{0, 1\}^{\leq 2l(k)}$  while  $h()$  is in  $\{0, 1\}^{l(k)}$ . Therefore, the inequality,  $2l(k) > l(k)$ , allows us to regard all of them  $h_i()$  for simplicity. Let  $h()$  denote a real world hash function. For practical recoveries of random oracles in the real world, we define;  $h_1(x) = h(00|x|00)$ ,  $h_2(x) = h(01|x|01)$ ,  $h_3(x) = h(01|x|10)$ ,  $h_4(x) = h(10|x|10)$  and  $h_5(x) = h(11|x|11)$  by following the constructions given in the Bellare and Rogaway's work[3].  $|$  denotes the concatenation.

NUMERICAL ASSUMPTION. We assume that all numerical operations of the protocol are on the cyclic group where it is hard to solve the Diffie-Hellman problem as well as the discrete logarithm problem. We consider the multiplicative group  $Z_p^*$  and actually use its prime-order subgroup  $Z_q$ . Note that we should use only its main operation, a modular multiplication, for easy generalization. For the purpose, *Bob* chooses  $g$  which generates a prime-order subgroup  $Z_q$  where  $p = qr + 1$ . Note that a prime  $q$  must be sufficiently large ( $> l(k)$ ) to resist Pohlig-Hellman decomposition and various index-calculus methods but much smaller than  $p$ [31, 33, 34]. It is easy to make  $g$  by  $\alpha^{(p-1)/q}$  where  $\alpha$  generates  $Z_p^*$ .  $Z_p^*$  is also applicable but  $Z_q$  is preferred for efficiency and for preventing a small subgroup confinement more effectively. By confining all exponentiation to the large prime-order subgroup through  $g$  of  $Z_q$ , each part of the protocol is able to detect on-line attack whenever a received exponential is confined to a small subgroup. We can use a secure prime modulus  $p$  such that  $(p-1)/2q$  is also prime or each prime factor of  $(p-1)/2q$  is larger than  $q$ , or a safe prime modulus  $p$  such that  $p = 2q + 1$ [25]. However, we strongly recommend to use a secure prime modulus  $p$ . Such a modulus should make our modified protocol secure and efficient.

## 2.2 Protocol Description

The followings are describing the setup and the run of our protocol.

PROTOCOL SETUP. This step determines and publishes global parameters of AMP.

1. *Alice* and *Bob* share  $g$ ,  $p$  and  $q$ .
2. *Alice* chooses  $\pi \in_R \{0, 1\}^{\omega(k)}$  and notify *Bob*, in an authentic and confidential manner<sup>4</sup>.
3. *Bob* chooses  $\tau \in_R \{0, 1\}^{t(k)}$  and stores  $(id, \tau, \nu = g^v)$  where  $v = h_1(\tau, \pi)$ .
4.  $id$  indicates precisely a user name.

*Bob* should throw away  $\pi$  and  $v$  but keep  $\tau$  and  $\nu$ .

PROTOCOL RUN. Note that the cases,  $x \in \{0, 1\}^1$ ,  $y \in \{0, 1\}^1$ ,  $\mathcal{G}_1 \in \{0, 1\}^1$ ,  $\mathcal{G}_2 \in \{0, 1\}^1$ ,  $\nu \in \{0, 1\}^1$ , and their small subgroup confinements must be avoided for a security reason.

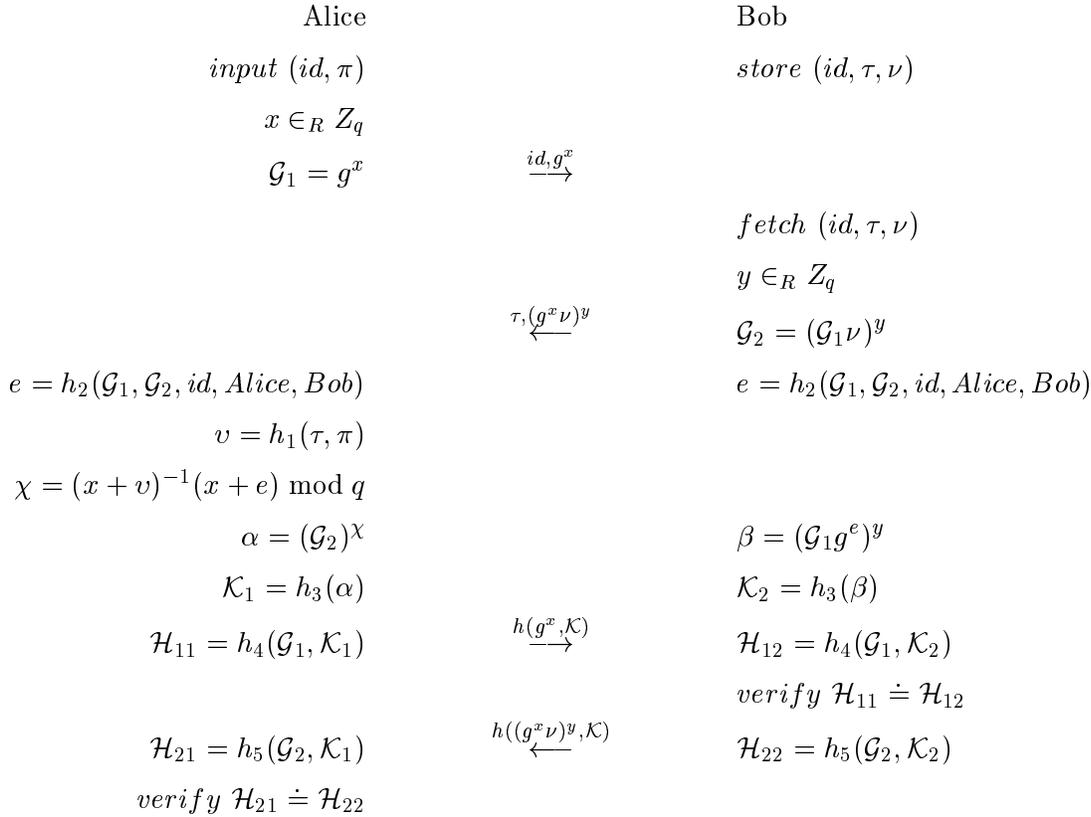


Figure 1. AMP Protocol

---

<sup>4</sup>We imply by “an authentic and confidential manner” that the corresponding parameters are shared by some safe method, for example, secure registration, off-line distribution with picture id proof.

The following steps explain how the protocol is executed in Figure 1.

1. *Alice* computes  $\mathcal{G}_1 = g^x$  by choosing  $x \in_R Z_q$  and sends  $(id, \mathcal{G}_1)$  to *Bob*.
2. After receiving message 1, *Bob* loads  $\tau$  and  $\nu$ , and computes  $\mathcal{G}_2 = (\mathcal{G}_1\nu)^y$  by choosing  $y \in_R Z_q$ . This can be done by the simultaneous exponentiation method. Note that  $\mathcal{G}_2 = (g^x)^y(\nu)^y = (g^x\nu)^y = g^{(x+v)y}$ . He sends  $(\tau, \mathcal{G}_2)$  to *Alice*.
3. After receiving message 2, *Alice* computes  $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $v = h_1(\tau, \pi)$ ,  $\chi = (x + v)^{-1}(x + e) \bmod q$  and  $\alpha = (\mathcal{G}_2)^\chi$ . Note that  $\alpha = (g^{(x+v)y})^{(x+v)^{-1}(x+e)} = g^{y(x+e)}$ . She computes  $\mathcal{K}_1 = h_3(\alpha)$  and  $\mathcal{H}_{11} = h_4(\mathcal{G}_1, \mathcal{K}_1)$ . She sends  $\mathcal{H}_{11}$  to *Bob*.
4. While waiting for message 3, *Bob* computes  $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $\beta = (g^x)^y g^{ey} = (g^x g^e)^y = g^{(x+e)y}$ ,  $\mathcal{K}_2 = h_3(\beta)$  and  $\mathcal{H}_{12} = h_4(\mathcal{G}_1, \mathcal{K}_2)$ . After receiving message 3, *Bob* compares  $\mathcal{H}_{12}$  with  $\mathcal{H}_{11}$ . If they are matched, then he computes  $\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$  and sends  $\mathcal{H}_{22}$  to *Alice*. This means he authenticated *Alice* who knows  $v$  (actually,  $\pi$ ), and agreed upon  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$ .
5. While waiting for message 4 from *Bob*, *Alice* computes  $\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1)$ . After receiving message 4, she compares  $\mathcal{H}_{21}$  with  $\mathcal{H}_{22}$ . If they are matched, *Alice* also agrees on  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$  with authenticating *Bob* who knows  $\nu$ .

### 2.3 Small Discussion

AMP passes four messages between *Alice* and *Bob* and agrees on  $g^{(x+e)y}$  rather than  $g^{xy}$ . The existence of  $e$  is explicit because of withstanding *Eve* who stole  $\nu$ , i.e., the password file compromise. For example, if *Eve* knowing  $\nu$  sends  $\nu^x$  to *Bob*, then *Bob* will reply with  $(\nu^x\nu)^y$  and compute  $\mathcal{K}_2 = (\nu^x g^e)^y = g^{(v+x+e)y}$ . However, *Eve* has only  $\nu$ ,  $x$ ,  $e$  and  $\nu^y (= \mathcal{G}_2^{(x+1)^{-1}})$  so that she still cannot find the key without knowing  $v$ . Regarding the benefits from the simultaneous exponentiation method, each party computes the exponentiation,  $O((\log n)^3)$ , only for two times, respectively. Final two steps can be modified, for example,  $\mathcal{H}_{11} = h_4(\alpha, \mathcal{G}_1, \mathcal{G}_2, id, A, B)$  and  $\mathcal{H}_{22} = h_5(\beta, \mathcal{G}_2, \mathcal{G}_1, B, A, id)$ . As we mentioned in section 2.1, we can remove  $\tau$  from message 2 for implicit salt (we show such a variant in the next section). For updating the existing system such as a Unix password file, we can modify  $v$  such that  $v = h_1(\tau, h(\tau, \pi))$  where  $h(\tau, \pi)$  is an existing verifier. We supposed  $\varphi()$  in  $\{0, 1\}^{\leq 2l(k)}$  as  $h_1()$  in  $\{0, 1\}^{l(k)}$ . For the recovery in real application, recommending SHA-1 or RIPEMD-160 for 160-bit hash, we defined;  $h_1(x) = h(00|x|00)$ . However, if other functions of 128-bit hash are used, we define  $\varphi(x) = h(00|x|00|x)h(10|x|10|x)$  instead of  $h_1(x)$ .

## 3 Extended Protocols

We present four explicit variants of AMP. They are called AMP<sup>i</sup>, AMP<sup>n</sup>, AMP<sup>+</sup>, AMP<sup>++</sup>. AMP<sup>i</sup> is extended for accommodating implicit salt while AMP<sup>n</sup> is extended for non-verifier au-

thentication similar to clear-text password authentication.  $\text{AMP}^+$  and  $\text{AMP}^{++}$  are extended for avoiding the information leakage arguments though such are not critical in  $\text{AMP}$ [24].

### 3.1 $\text{AMP}^i$

$\text{AMP}^i$  is a simple extension of  $\text{AMP}$  for accommodating implicit salt.

PROTOCOL SETUP. This step determines and publishes global parameters of  $\text{AMP}^i$ .

1. *Alice* and *Bob* share  $g$ ,  $p$  and  $q$ .
2. *Alice* chooses  $\pi \in_R \{0, 1\}^{\omega(k)}$  and notify *Bob*, in an authentic and confidential manner.
3. *Bob* stores  $(id, \nu = g^\nu)$  where  $\nu = h_1(id, B, \pi)$ .
4.  $id$  indicates precisely a user name.

*Bob* should throw away  $\pi$  and  $\nu$  but keep  $\nu$ .

PROTOCOL RUN. Note that the cases,  $x \in \{0, 1\}^1$ ,  $y \in \{0, 1\}^1$ ,  $\mathcal{G}_1 \in \{0, 1\}^1$ ,  $\mathcal{G}_2 \in \{0, 1\}^1$ ,  $\nu \in \{0, 1\}^1$ , and their small subgroup confinements must be avoided for a security reason.

Alice		Bob
<i>input</i> $(id, \pi)$		<i>store</i> $(id, \nu)$
$x \in_R Z_q$		
$\mathcal{G}_1 = g^x$	$\xrightarrow{id, g^x}$	
$\nu = h_1(id, B, \pi)$		<i>fetch</i> $(id, \nu)$
$\xi = (x + \nu)^{-1} \bmod q$	$\xleftarrow{(g^x \nu)^y}$	$y \in_R Z_q$
$e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$		$\mathcal{G}_2 = (\mathcal{G}_1 \nu)^y$
$\chi = \xi(x + e) \bmod q$		$e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$
$\alpha = (\mathcal{G}_2)^\chi$		$\beta = (\mathcal{G}_1 g^e)^y$
$\mathcal{K}_1 = h_3(\alpha)$		$\mathcal{K}_2 = h_3(\beta)$
$\mathcal{H}_{11} = h_4(\mathcal{G}_1, \mathcal{K}_1)$	$\xrightarrow{h(g^x, \mathcal{K})}$	$\mathcal{H}_{12} = h_4(\mathcal{G}_1, \mathcal{K}_2)$
$\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1)$		<i>verify</i> $\mathcal{H}_{11} \doteq \mathcal{H}_{12}$
<i>verify</i> $\mathcal{H}_{21} \doteq \mathcal{H}_{22}$	$\xleftarrow{h((g^x \nu)^y, \mathcal{K})}$	$\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$

Figure 2.  $\text{AMP}^i$  Protocol

The following steps explain how the protocol is executed in Figure 2.

1. *Alice* computes  $\mathcal{G}_1 = g^x$  by choosing  $x \in_R \mathbb{Z}_q$  and sends  $(id, \mathcal{G}_1)$  to *Bob*.
2. After receiving message 1, *Bob* loads  $\nu$ , and computes  $\mathcal{G}_2 = (\mathcal{G}_1 \nu)^y$  by choosing  $y \in_R \mathbb{Z}_q$ . This can be done by the simultaneous exponentiation method. Note that  $\mathcal{G}_2 = (g^x)^y (\nu)^y = (g^x \nu)^y = g^{(x+\nu)y}$ . He sends  $\mathcal{G}_2$  to *Alice*.
3. While waiting for message 2, *Alice* computes  $v = h_1(id, B, \pi)$  and  $\xi = (x + v)^{-1} \bmod q$ . After receiving message 2, *Alice* computes  $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $\chi = \xi(x + e) \bmod q$  and  $\alpha = (\mathcal{G}_2)^\chi$ . Note that  $\alpha = (g^{(x+\nu)y})^{(x+\nu)^{-1}(x+e)} = g^{y(x+e)}$ . She computes  $\mathcal{K}_1 = h_3(\alpha)$  and  $\mathcal{H}_{11} = h_4(\mathcal{G}_1, \mathcal{K}_1)$ . She sends  $\mathcal{H}_{11}$  to *Bob*.
4. While waiting for message 3, *Bob* computes  $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $\beta = (g^x)^y g^{ey} = (g^x g^e)^y = g^{(x+e)y}$ ,  $\mathcal{K}_2 = h_3(\beta)$  and  $\mathcal{H}_{12} = h_4(\mathcal{G}_1, \mathcal{K}_2)$ . After receiving message 3, *Bob* compares  $\mathcal{H}_{12}$  with  $\mathcal{H}_{11}$ . If they are matched, then he computes  $\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$  and sends  $\mathcal{H}_{22}$  to *Alice*. This means he authenticated *Alice* who knows  $v$  (actually,  $\pi$ ), and agreed upon  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$ .
5. While waiting for message 4 from *Bob*, *Alice* computes  $\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1)$ . After receiving message 4, she compares  $\mathcal{H}_{21}$  with  $\mathcal{H}_{22}$ . If they are matched, *Alice* also agrees on  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$  with authenticating *Bob* who knows  $\nu$ .

AMP<sup>i</sup> reduces its running time by allowing *Alice* to compute mod  $q$  inverse while waiting for message 2 from *Bob*. This is due to the use of implicit salt.

### 3.2 AMP<sup>n</sup>

AMP<sup>n</sup> is a simple extension of AMP for accommodating non-verifier authentication similar to clear-text password authentication. Its assumption is that the protocol is vulnerable to the password file compromise as like EKE. The goal of this extension is only for maximizing efficiency not for security. Actually, AMP<sup>n</sup> is the most efficient but more vulnerable compared to other AMP's (as like comparing EKE and A-EKE). So we call AMP<sup>n</sup> "AMP-naked" because it is not protected by  $e$  and salted  $\nu$ .

PROTOCOL SETUP. This step determines and publishes global parameters of AMP<sup>n</sup>.

1. *Alice* and *Bob* share  $g$ ,  $p$  and  $q$ .
2. *Alice* chooses  $\pi \in_R \{0, 1\}^{\omega(k)}$  and notify *Bob*, in an authentic and confidential manner.
3. *Bob* stores  $(id, \nu = g^\pi)$  where  $v = h_1(\pi)$ .
4.  $id$  indicates precisely a user name.

Bob should throw away  $\pi$  and  $v$  but keep  $\nu$ .

PROTOCOL RUN. Note that the cases,  $x \in \{0, 1\}^1$ ,  $y \in \{0, 1\}^1$ ,  $\mathcal{G}_1 \in \{0, 1\}^1$ ,  $\mathcal{G}_2 \in \{0, 1\}^1$ ,  $\nu \in \{0, 1\}^1$ , and their small subgroup confinements must be avoided for a security reason.

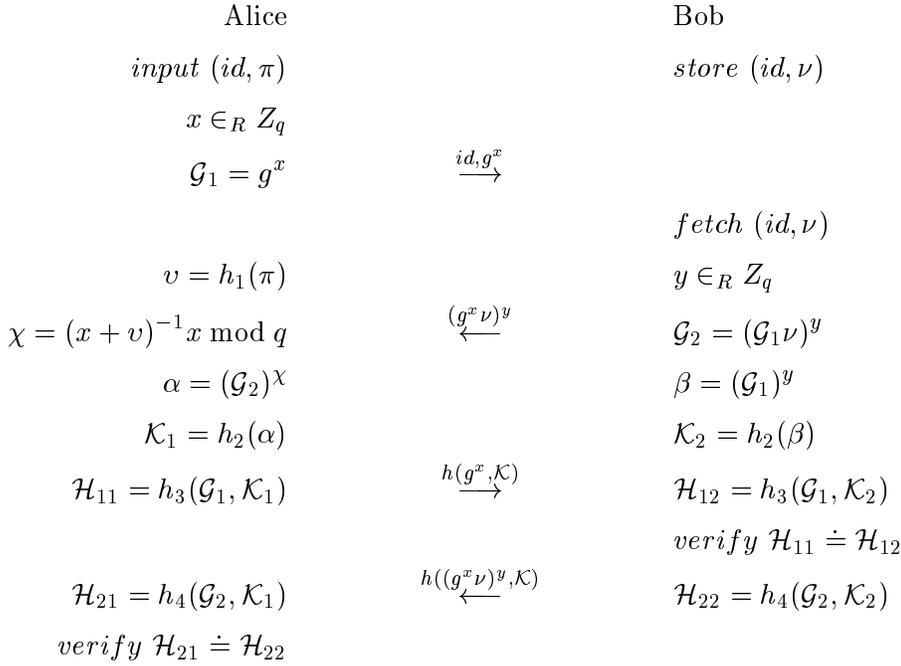


Figure 3. AMP<sup>n</sup> Protocol

The following steps explain how the protocol is executed in Figure 3.

1. *Alice* computes  $\mathcal{G}_1 = g^x$  by choosing  $x \in_R Z_q$  and sends  $(id, \mathcal{G}_1)$  to *Bob*.
2. After receiving message 1, *Bob* loads  $\nu$ , and computes  $\mathcal{G}_2 = (\mathcal{G}_1 \nu)^y$  by choosing  $y \in_R Z_q$ . This can be done by the simultaneous exponentiation method. Note that  $\mathcal{G}_2 = (g^x)^y (\nu)^y = (g^x \nu)^y = g^{(x+v)y}$ . He sends  $\mathcal{G}_2$  to *Alice*.
3. While waiting for message 2, *Alice* computes  $v = h_1(\pi)$  and  $\chi = (x+v)^{-1} x \bmod q$ . After receiving message 2, *Alice* computes  $\alpha = (\mathcal{G}_2)^\chi$ . Note that  $\alpha = (g^{(x+v)y})^{(x+v)^{-1} x} = g^{yx}$ . She computes  $\mathcal{K}_1 = h_2(\alpha)$  and  $\mathcal{H}_{11} = h_3(\mathcal{G}_1, \mathcal{K}_1)$ . She sends  $\mathcal{H}_{11}$  to *Bob*.
4. While waiting for message 3, *Bob* computes  $\beta = (g^x)^y g^y = (g^x)^y = g^{xy}$ ,  $\mathcal{K}_2 = h_2(\beta)$  and  $\mathcal{H}_{12} = h_3(\mathcal{G}_1, \mathcal{K}_2)$ . After receiving message 3, *Bob* compares  $\mathcal{H}_{12}$  with  $\mathcal{H}_{11}$ . If they are matched, then he computes  $\mathcal{H}_{22} = h_4(\mathcal{G}_2, \mathcal{K}_2)$  and sends  $\mathcal{H}_{22}$  to *Alice*. This means he authenticated *Alice* who knows  $v$  (actually,  $\nu$  rather than only  $\pi$ ), and agreed upon  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$ .

5. While waiting for message 4 from *Bob*, *Alice* computes  $\mathcal{H}_{21} = h_4(\mathcal{G}_2, \mathcal{K}_1)$ . After receiving message 4, she compares  $\mathcal{H}_{21}$  with  $\mathcal{H}_{22}$ . If they are matched, *Alice* also agrees on  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$  with authenticating *Bob* who knows  $\nu$ .

AMP<sup>n</sup>, so-called AMP-naked, extremely reduces its running time by allowing *Alice* to compute all mod  $q$  operations while waiting for message 2 from *Bob* and by allowing both parties not to compute  $e$ -related operations at all. This is due to the “naked-assumption” such that the protocol is vulnerable to the password file compromise. For example, if *Eve* who stole  $\nu$  sends  $\nu^x$  to *Bob*, she can complete the protocol because  $\mathcal{G}_2 = \nu^{(x+1)y}$  while  $\beta = \nu^{xy}$ [20]. However, AMP<sup>n</sup> provides much more well-defined form of authentication for having EKE-security.

### 3.3 AMP<sup>+</sup>

AMP<sup>+</sup> is a simple extension of AMP for avoiding information leakage arguments. Such arguments are simple (message and key are clearly unrelated?) and not harmful in AMP[24]. Protocol setup of AMP<sup>+</sup> is exactly same to that of AMP so that we skip it.

PROTOCOL RUN. Note that the cases,  $x \in \{0, 1\}^1$ ,  $y \in \{0, 1\}^1$ ,  $\mathcal{G}_1 \in \{0, 1\}^1$ ,  $\mathcal{G}_2 \in \{0, 1\}^1$ ,  $\nu \in \{0, 1\}^1$ , and their small subgroup confinements must be avoided for a security reason.

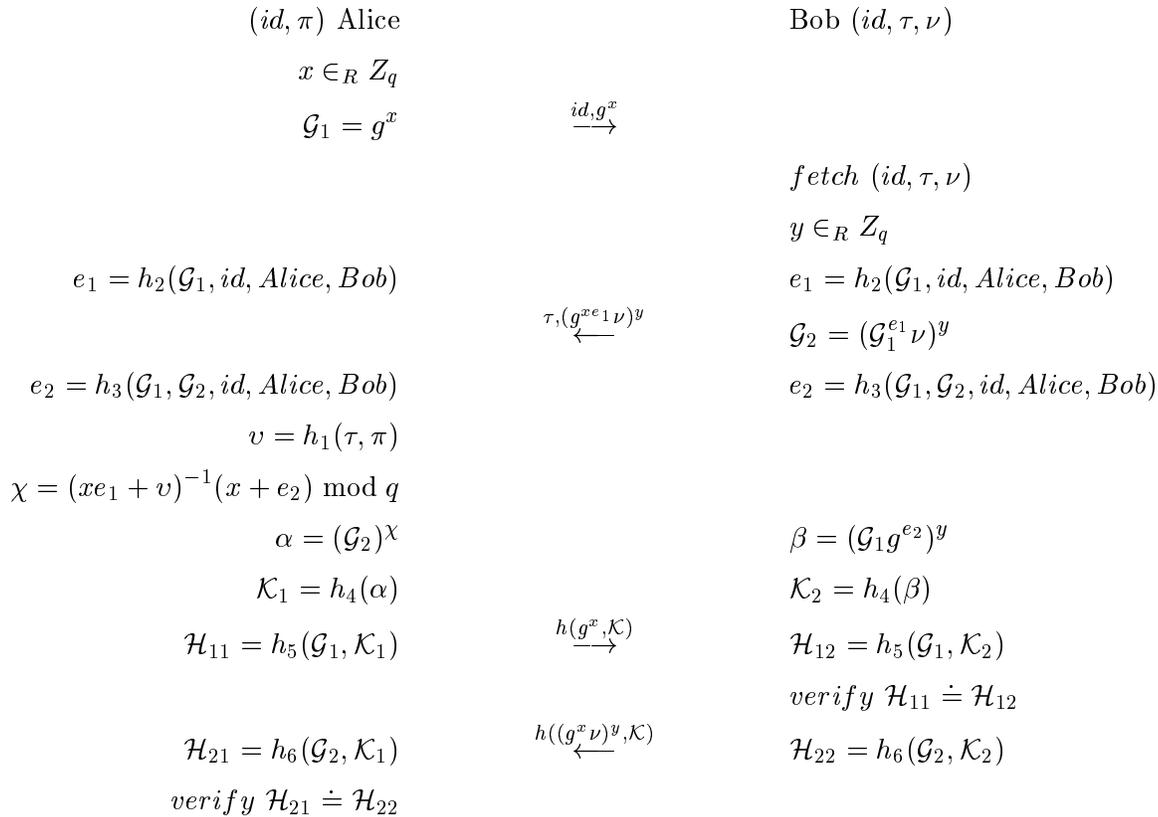


Figure 4. AMP<sup>+</sup> Protocol

The following steps explain how the protocol is executed in Figure 4.

1. *Alice* computes  $\mathcal{G}_1 = g^x$  by choosing  $x \in_R \mathbb{Z}_q$  and sends  $(id, \mathcal{G}_1)$  to *Bob*.
2. After receiving message 1, *Bob* loads  $\tau$  and  $\nu$ , and computes  $e_1 = h_2(\mathcal{G}_1, id, Alice, Bob)$ ,  $\mathcal{G}_2 = (\mathcal{G}_1^{e_1} \nu)^y$  by choosing  $y \in_R \mathbb{Z}_q$ . This can be done by the simultaneous exponentiation method. Note that  $\mathcal{G}_2 = (g^x)^{e_1 y} (\nu)^y = g^{(xe_1+v)y}$ . He sends  $(\tau, \mathcal{G}_2)$  to *Alice*.
3. While waiting message 2, *Alice* computes  $e_1 = h_2(\mathcal{G}_1, id, Alice, Bob)$ . After receiving message 2, *Alice* computes  $e_2 = h_3(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $v = h_1(\tau, \pi)$ ,  $\chi = (xe_1 + v)^{-1}(x + e_2) \bmod q$  and  $\alpha = (\mathcal{G}_2)^\chi$ . Note that  $\alpha = (g^{(xe_1+v)y})^{(xe_1+v)^{-1}(x+e_2)} = g^{y(x+e_2)}$ . She computes  $\mathcal{K}_1 = h_4(\alpha)$  and  $\mathcal{H}_{11} = h_5(\mathcal{G}_1, \mathcal{K}_1)$ . She sends  $\mathcal{H}_{11}$  to *Bob*.
4. While waiting for message 3, *Bob* computes  $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $\beta = (g^x)^y g^{e_2 y} = (g^x g^{e_2})^y = g^{(x+e_2)y}$ ,  $\mathcal{K}_2 = h_4(\beta)$  and  $\mathcal{H}_{12} = h_5(\mathcal{G}_1, \mathcal{K}_2)$ . After receiving message 3, *Bob* compares  $\mathcal{H}_{12}$  with  $\mathcal{H}_{11}$ . If they are matched, then he computes  $\mathcal{H}_{22} = h_6(\mathcal{G}_2, \mathcal{K}_2)$  and sends  $\mathcal{H}_{22}$  to *Alice*. This means he authenticated *Alice* who knows  $v$  (actually,  $\pi$ ), and agreed upon  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$ .
5. While waiting for message 4 from *Bob*, *Alice* computes  $\mathcal{H}_{21} = h_6(\mathcal{G}_2, \mathcal{K}_1)$ . After receiving message 4, she compares  $\mathcal{H}_{21}$  with  $\mathcal{H}_{22}$ . If they are matched, *Alice* also agrees on  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$  with authenticating *Bob* who knows  $\nu$ .

We should define  $h_6(x) = h(10|x|01)$  for  $\text{AMP}^+$ . The randomness of  $e_1$  is totally dependent upon the randomness of  $g^x$  so that *Bob* cannot contribute to its randomness, while the randomness of  $e_2$  is dependent upon the randomness of  $\mathcal{G}_2$  as well as  $g^x$ . The information leakage argument has been clearly avoided since  $\mathcal{G}_2 = g^{(xe_1+v)y}$  while the agreed key is  $g^{(x+e_2)y}$ . They have different structures so that  $\mathcal{G}_2$  is not related to the agreed key regarding the intractability of the discrete logarithm problem and the Diffie-Hellman problem.

### 3.4 $\text{AMP}^{++}$

$\text{AMP}^+$  is the second extension of  $\text{AMP}$  for avoiding information leakage arguments.

**PROTOCOL SETUP.** This step determines and publishes global parameters of  $\text{AMP}^{++}$ .

1. *Alice* and *Bob* shares  $g, p$  and  $q$ .
2. *Alice* chooses  $\pi \in_R \{0, 1\}^{\omega(k)}$  and notify *Bob*, in an authentic manner.
3.  $id$  indicates an identifier or name of *Alice*; more precisely a user name.
4. *Bob* stores  $(id, \nu = g^{-v})$  where  $v = h_1(id, Bob, \pi)$ <sup>5</sup>.

---

<sup>5</sup>We can also use the explicit salt scheme such that  $v = h_1(\tau, \pi)$  where  $\tau \in_R \{0, 1\}^{\iota(k)}$ . See later part.

Bob should throw away  $\pi$  and  $v$  but keep  $id$  and  $\nu$ .

PROTOCOL RUN. The following describes how to run AMP<sup>++</sup>. Note that the cases,  $x_1 \in \{0, 1\}^1$ ,  $x_2 \in \{0, 1\}^1$ ,  $y \in \{0, 1\}^1$ ,  $\mathcal{G}_0 \in \{0, 1\}^1$ ,  $\mathcal{G}_1 \in \{0, 1\}^1$ ,  $\mathcal{G}_2 \in \{0, 1\}^1$ ,  $\nu \in \{0, 1\}^1$ , and their small subgroup confinements must be avoided for a security reason.

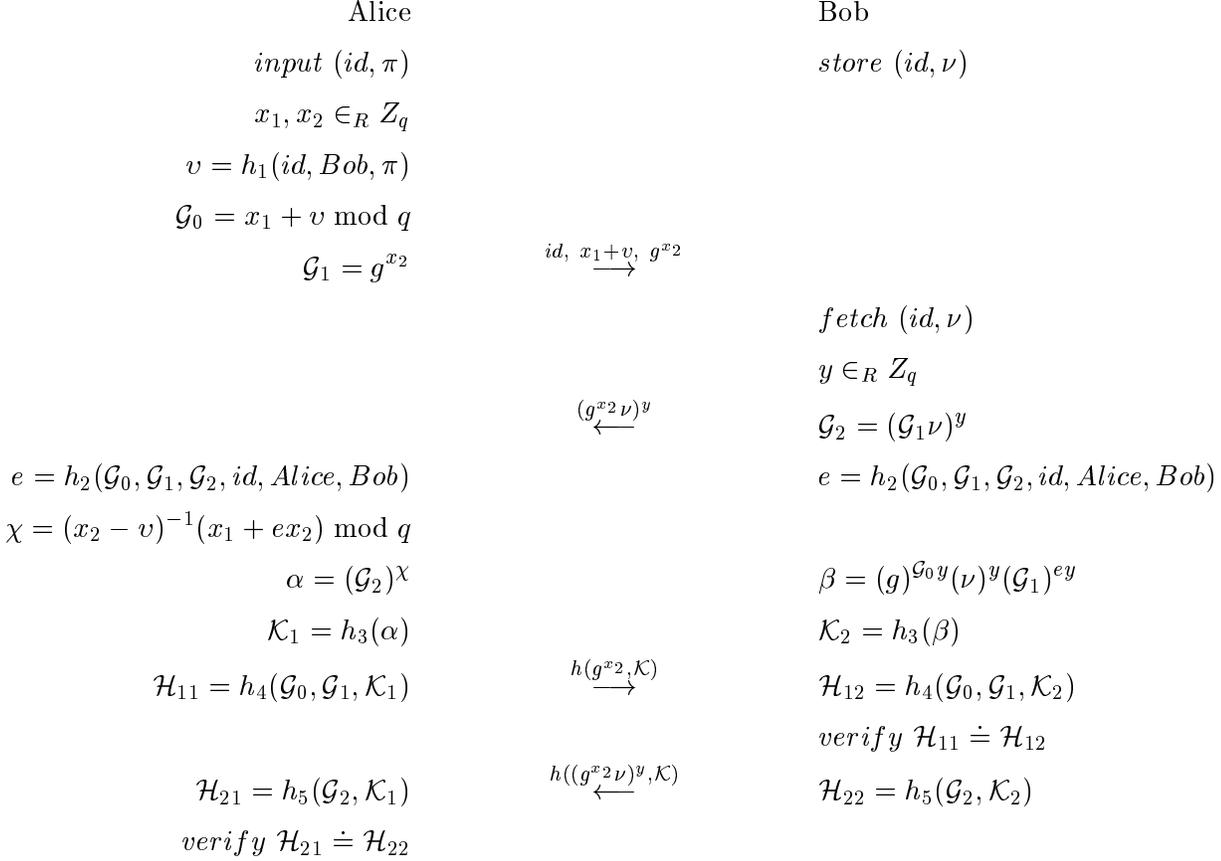


Figure 5. AMP<sup>++</sup> Protocol

The following steps describe how the protocol is executed in Figure 5.

1. *Alice* computes  $v = h_1(id, Bob, \pi)$ ,  $\mathcal{G}_0 = x_1 + v \bmod q$ , and  $\mathcal{G}_1 = g^{x_2}$  by choosing  $x_1, x_2 \in_R Z_q$  and sends  $(id, \mathcal{G}_0, \mathcal{G}_1)$  to *Bob*.
2. After receiving message 1, *Bob* loads  $\nu$ , and computes  $\mathcal{G}_2 = (\mathcal{G}_1\nu)^y$  by choosing  $y \in_R Z_q$ . This can be done by the simultaneous exponentiation method, i.e.,  $\mathcal{G}_1^y\nu^y$ . Note that  $\mathcal{G}_2 = (g^{x_2\nu})^y = g^{(x_2-v)y}$ . He sends  $\mathcal{G}_2$  to *Alice*.
3. After receiving message 2, *Alice* computes  $e = h_2(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $\chi = (x_2 - v)^{-1}(x_1 + ex_2) \bmod q$ , and  $\alpha = (\mathcal{G}_2)^\chi$ . Note that  $\alpha = (g^{(x_2-v)y})^{(x_2-v)^{-1}(x_1+ex_2)} = g^{y(x_1+ex_2)}$ . She computes  $\mathcal{K}_1 = h_3(\alpha)$  and  $\mathcal{H}_{11} = h_4(\mathcal{G}_0, \mathcal{G}_1, \mathcal{K}_1)$ . She sends  $\mathcal{H}_{11}$  to *Bob*.

4. While waiting for message 3, *Bob* computes  $e = h_2(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$ ,  $\beta = (g)^{\mathcal{G}_0 y} (\nu)^y (\mathcal{G}_1)^{e y}$ ,  $\mathcal{K}_2 = h_3(\beta)$  and  $\mathcal{H}_{12} = h_4(\mathcal{G}_0, \mathcal{G}_1, \mathcal{K}_2)$ . Note  $\beta = g^{(x_1+v)y} g^{-vy} g^{x_2 e y} = g^{(x_1+ex_2)y}$ . After receiving message 3, *Bob* compares  $\mathcal{H}_{11}$  with  $\mathcal{H}_{12}$ . If they are equal to each other, then he computes  $\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$  and sends it to *Alice*. This means he authenticated *Alice* who knows  $v$  (actually,  $\pi$ ), and agreed upon  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$ .
5. While waiting for message 4, *Alice* computes  $\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1)$ . After receiving message 4, she compares  $\mathcal{H}_{12}$  with  $\mathcal{H}_{22}$ . If  $\mathcal{H}_{12} = \mathcal{H}_{22}$ , *Alice* also agrees on  $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$  with authenticating *Bob* who knows  $\nu$ .

AMP<sup>++</sup> also avoids the information leakage argument since the agreed key is  $g^{(x_1+ex_2)y}$  while  $\mathcal{G}_2 = g^{(x_2-v)y}$ . Note that  $\alpha^a \beta^b \gamma^c$  needs 25% more multi-precision multiplications than  $\alpha^a$  does on the average through the simultaneous multiple exponentiation method[36, 29]. Considering the benefit of the simultaneous method, the parallel exponentiation is still three times ( $3E$ ) due to the use of implicit salt.

EXPLICIT SALT IN AMP<sup>++</sup>. For efficiency ( $3E$ ), we considered the implicit salt scheme. However, we can accommodate the conventional salt scheme at the cost of parallel exponentiation ( $4E$ ). For using explicit salt, *Alice* should compute  $v$  after receiving message 2 so that she could compute and pass  $\mathcal{G}_0$  with  $\mathcal{H}_1$  in step 3. Therefore, *Bob* is able to compute  $\beta$  after receiving message 3. That is, the protocol loses the parallel computation of  $\alpha$  and  $\beta$  so that the parallel exponentiation cost is to be  $4E$  rather than  $3E$ . This is the worst case in our AMP family.

## 4 Analysis and Comparison

This section examines the security against conventional attacks and then discusses the advantages and disadvantages of AMP in terms of security, efficiency and constraints, while comparing AMP to other functionally-similar (secure) protocols such as A-EKE, B-SPEKE, SRP, GXY, SNAPL-X, AuthA and PAK-X[8, 19, 40, 28, 6, 10]. AMP implies the original protocol and other extensions if we do not explicitly notify.

### 4.1 Security of AMP

1. AMP provides perfect forward secrecy via the Diffie-Hellman problem. That is, even if  $\pi$  (or  $v$ ) is compromised, *Eve* cannot find old session keys because she is not able to solve the Diffie-Hellman problem.
2. Denning-Sacco attack is the case that *Eve*, who compromised an old session key, attempts to find  $\pi$  or to make the oracle accept her[11]. For the purpose, *Eve* also has to solve the Diffie-Hellman problem. Moreover, the actual session key is computed

by  $\mathcal{K} = h_2(g^{xy})$ . Therefore, a compromised old session key is not helpful for *Eve* in attempting the Denning-Sacco attack.

3. Replay attack is negligible because  $\mathcal{G}_1$  should include an ephemeral parameter of *Alice* while the others such as  $\mathcal{G}_2$ ,  $\mathcal{H}_1$  and  $\mathcal{H}_2$ , should include ephemeral parameters of both parties of the session. Finding those parameters is similar to solving the discrete logarithm problem and each parameter is bounded by  $2^{-l(k)} < 2^{-k}$ . Therefore, both active replay and succeeding verification are negligible.
4. Small subgroup confinement is defeated and avoided by confining to the large prime-order subgroup. An intentional small subgroup confinement can be detected easily.
5. On-line guessing attack is detectable and the following off-line analysis can be frustrated, even if *Eve* attempts to disguise parties. Impersonation of the party or man-in-the-middle attack is also infeasible without knowing  $v$  or  $\nu$ . Actually, *Eve* is able to perform the on-line attack in either side such as *Alice* or *Bob* but such an attack is detectable.
6. Off-line guessing attack is also infeasible because *Eve* cannot disintegrate  $\mathcal{G}_2$ . Partition attack is to reduce the set of passwords logarithmically by asking the oracle in parallel with off-line analysis, while chosen exponent attack is to analyze it via her chosen exponent. Both attacks are infeasible because *Eve* cannot solve or reduce  $y' = (x + v)y(x + v')^{-1}$ .
7. Security against password-file compromise is the basic property of AMP except AMP<sup>n</sup> that has a naked assumption.
8. Information leakage is not critical in AMP by virtue of random oracles for  $v$  and  $e$ , though AMP<sup>+</sup> and AMP<sup>++</sup> were proposed for clearance of related arguments.

## 4.2 Efficiency and Constraints

Performance of these protocol families can be approximated in terms of communication and computation loads (see Table 1). We summarize the efficiency and constraints of AMP.

EFFICIENCY. The efficiency of AMP can be discussed as follows.

1. In the aspect of a communication load, AMP has only four protocol steps while the number of large message blocks is only two in AMP. They are  $\mathcal{G}_1$  and  $\mathcal{G}_2$ . For AMP<sup>++</sup>, the size of  $\mathcal{G}_0$  can be bounded by  $l(k) + \epsilon$  with negligible  $\epsilon$  when we use a secure prime.
2. A total amount of execution time could be approximated by the number of modular exponentiation by considering the parallel execution of both parties. We describe it as  $E(\text{Alice} : \text{Bob})$ . Note that AMP has intrinsically only  $3E$ , except that AMP<sup>++</sup> has  $4E$  with explicit salt. AMP has  $E((g)^x : -)$ ,  $E(- : (\mathcal{G}_1^y \nu^y))$  and  $E((\mathcal{G}_2)^x : (\mathcal{G}_1)^y g^{ey})$  while

all variants except AMP<sup>++</sup> with explicit salt have similar operations. Here ‘-’ means no-exponentiation such as  $O((\log n)^3)$ .

3. Each party of AMP performs only two exponentiations, respectively. It is the same to the number in the Diffie-Hellman scheme though AMP needs some more operations for larger base,  $Z_q$  operation or simultaneous exponentiation.
4. For run time parameters, each party generates only one random number, respectively, in AMP and AMP<sup>+</sup>. *Alice* can reduce her run time exponentiations to only one and parallel exponentiations to only two, by pre-computation of  $g^x$ . AMP<sup>++</sup> needs *Alice* to generate two random numbers.
5.  $\mathcal{G}_2$  can benefit from the simultaneous exponentiation method as  $(\mathcal{G}_1)^{y\nu^y}$ .  $\beta$  of AMP<sup>++</sup> must benefit from the simultaneous exponentiation method. As we mentioned already,  $\alpha^a\beta^b$  needs 16% and  $\alpha^a\beta^b\gamma^c$  needs 25% more multi-precision multiplications than  $\alpha^a$  does on the average[36, 29].
6. In step 3, *Alice* should compute  $(y + v)^{-1}$  but only in the  $q$ -order subgroup. Modular inversion,  $O((\log q)^2)$ , could be negligible against modular exponentiation,  $O((\log p)^3)$ . Moreover, the size of  $q$  can be bounded by only  $l(k) + \epsilon$  with negligible  $\epsilon$  by virtue of a secure prime. Note that  $O(\log l(k)) \ll O(\log p)$ .
7. AMP uses the main group operation so that it is *easy-to-generalize* in any cyclic groups. Therefore, AMP can be easily implemented on the elliptic curve group. A generalization on such a group must be very useful for further efficiency of space and speed, though there may a patent restriction.

A rigorous comparison to the other protocols is made in section 4.3

CONSTRAINTS. AMP gives very light constraints as follows.

1. AMP prefers  $g$  to be a generator of the large ( $\geq l(k)$ ) prime-order subgroup  $Z_q$  for defeating and avoiding a small subgroup confinement effectively by confining exponentials into the large prime-order subgroup[31]. A secure prime modulus is highly recommended for easy detection of an intentional small subgroup confinement and great efficiency of the protocols though a safe prime modulus is also favorable. Note that the secure prime is easier to get than the safe prime[25].
2. A compromise of  $\nu$  allows a guessing attack or a server impersonation but it is an inevitable feature of all verifier-based protocols[40].
3. AMP needs both parties to count the other side’s on-line failure to detect the on-line guessing attack. However, this is the shared requirement of all password protocols.

	<i>Protocol</i>	<i>Large</i>	<i>Exponentiations</i>			<i>Random Numbers</i>	
	<i>Steps</i>	<i>Blocks</i>	<i>Client</i>	<i>Server</i>	<i>Parallel</i>	<i>Client</i>	<i>Server</i>
A-EKE	7 (+4)	3 (+1)	4 (+2)	4 (+2)	6 (+3)	<b>1</b> (+0)	<b>1</b> (+0)
B-SPEKE	4 (+1)	3 (+1)	3 (+1)	4 (+2)	6 (+3)	<b>1</b> (+0)	2 (+1)
SRP	4 (+1)	<b>2</b> (+0)	3 (+1)	3 (+1)	4 (+1)	<b>1</b> (+0)	<b>1</b> (+0)
GXY	4 (+1)	<b>2</b> (+0)	4 (+2)	3 (+1)	5 (+2)	<b>1</b> (+0)	<b>1</b> (+0)
SNAPI-X	5 (+2)	5 (+3)	5 (+3)	4 (+2)	7 (+4)	2 (+1)	3 (+2)
AuthA	<b>3</b> (+0)	<b>2</b> (+0)	4 (+2)	3 (+1)	6 (+3)	<b>1</b> (+0)	<b>1</b> (+0)
PAK-X	<b>3</b> (+0)	3 (+1)	4 (+2)	4 (+2)	8 (+5)	<b>1</b> (+0)	2 (+1)
AMP	4 (+1)	<b>2</b> (+0)	<b>2</b> (+0)	<b>2</b> (+0)	<b>3</b> (+0)	<b>1</b> (+0)	<b>1</b> (+0)

Table 1: Comparisons of Verifier-based Protocols

### 4.3 Comparisons to Others

AMP is rigorously compared to the existing verifier-based protocols such as A-EKE, B-SPEKE, SRP, GXY, SNAPI-X, AuthA and PAK-X[8, 19, 40, 22, 28, 6, 10]. We disregard the security issue because all of them are believed secure.

Table 1 compares them with regard to several efficiency factors such as the number of protocol steps, large message blocks, and exponentiations. The random number is given as a subsidiary reference. Protocol steps and large blocks are critical factors to the communication load, while exponentiations and random numbers are to the computation load. The number of parallel exponentiations could compare approximately the amount of protocol execution time. The large block means a large cryptographic block based on the public-key cryptography such as the Diffie-Hellman exponential or the RSA message. The value in parenthesis implies the difference from the most efficient one that is denoted by bold characters.

As we can see in Table 1, AMP can be expected as the most efficient verifier-based protocol because it has the minimum values mostly. Note that  $AMP^i$  and  $AMP^n$  are a little more efficient than AMP while  $AMP^+$  is a little less efficient than AMP. Also note that  $AMP^{++}$  adds one more random number (and one more parallel exponentiation for explicit salt). Let us review the other related-protocols and discuss the superiority of AMP. Comparisons of  $AMP^n$  to the related protocols such as EKE, SPEKE, SNAPI, PAK[7, 18, 28, 10] under their naked assumption, are skipped in this document. However, the comparison result must be also explicit.

A-EKE. Bellare and Merritt introduced the first verifier-based protocol, A-EKE in 1993[8]. In A-EKE, the symmetric encryption required  $p$  very close to  $2^n$  where  $n$  is a bit-length of  $p$ , and random data to be padded to fill the block size[7]. A certified generator[32] and a digital signature scheme such as ElGamal [14] or p-NEW signature[30] were hard requirements.

A-EKE has many protocol steps and exponentiations, and requires a safe prime modulus.

B-SPEKE. Jablon proposed B-SPEKE, the verifier-based augmentation of SPEKE, in 1997[18, 19]. A-SPEKE was an A-EKE based extension of SPEKE while B-extension was better than A-extension[19]. Among the set of B-SPEKE, the combined B-SPEKE was the most optimized so that it is compared in Table 1. However, it required relatively heavy (parallel) exponentiations, and needed a server to choose another random number and a safe prime modulus. The parallel B-SPEKE allowed four parallel exponentiations but with requiring six message exchanges[19].

SRP. Wu proposed SRP that was notable in its practical approach, in 1998[40]. The benchmark showed its superiority to the earlier schemes[40]. Actually, SRP had been the most efficient verifier-based protocol before AMP was developed. SRP did not agree to the Diffie-Hellman exponential and was not favorable to the generalization, especially in the elliptic curve group. However, we can say SRP is a reasonably efficient and reliable verifier-based protocol (see Figure 6).

GXY. Kwon and Song proposed GXY that was derived from SRP but agreed to the Diffie-Hellman exponential, in 1999[22]. However, such an agreement was not critical and the protocol was actually less efficient than SRP. GXY was also not favorable to the generalization. However, GXY has been the base of studying and developing AMP family.

SNAPI-X. MacKenzie and Swaminathan introduced the first provable verifier-based protocol, SNAPI-X, in 1999[28]. It was the augmented version of their basic protocol SNAPI. It is notable in its full proof of security by combining RSA and Diffie-Hellman scheme. However, it has the worst results for most factors in Table 1, and has additional constraints on choosing specific parameters and export/patent restrictions.

AUTHA. Bellare and Rogaway introduced a provable approach, AuthA, by assuming an ideal cipher in 2000[6]. AuthA, as an extended version of the famous DH-EKE, well defines related attributes formally. It reduces her protocol steps without exchanging salt but must add two more steps to accommodate the existing salt schemes. Coincidentally, its exponentiation sequences and contents are almost the same to GXY[22]. Table 1 also shows such a result.

PAK-X. Boyko, MacKenzie, and Patel introduced the provable verifier-based protocol, PAK-X, in 2000[10]. PAK-X is notable in its clearly provable approach. PAK-X is the last protocol in their three kinds of protocols but the only one that uses a verifier. It also reduces the number of protocol steps without exchanging salt. But instead, it is ranked as the worst performance protocol regarding the parallel exponentiations in Table 1.

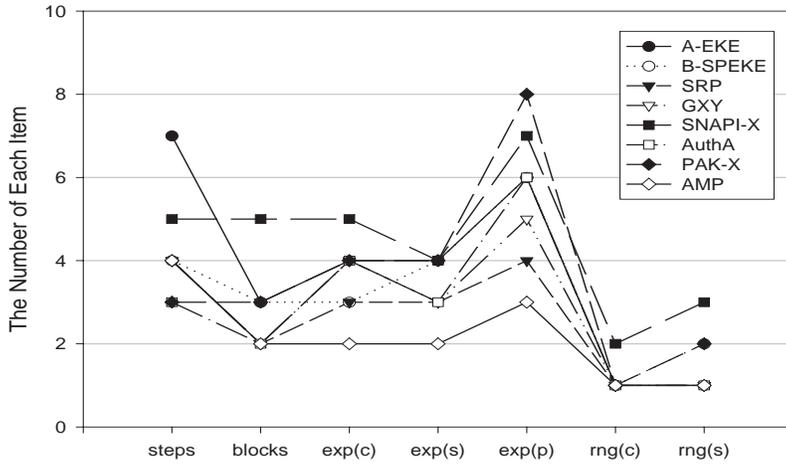


Figure 6. Graphical Representation of Table 1

WHY AMP. Figure 6 rewrites Table 1 graphically so that we can see AMP( $\diamond$ ) has the best performance. The following items summarize well why AMP is believed the best in this paper.

1. AMP is a secure verifier-based protocol<sup>6</sup> and it is provable in the random oracle model.
2. AMP is the most efficient protocol among the existing verifier-based protocols.
3. AMP has the light constraints and is easy to generalize, e.g., in elliptic curve groups for further efficiency.
4. AMP has several variants for accomodating implicit verifier or for pure password-based authentication.
5. AMP truly allows the Diffie-Hellman based key agreement; (1) *Alice* sends  $g^x$  to *Bob* who simply raises it to  $y$  with random number  $e$ , i.e.  $g^{(x+e)y}$ ; (2) *Bob* sends  $g^y$  to *Alice* by hiding it as  $(g^y)^{(x+v)}$  while *Alice* obtains it by  $((g^y)^{(x+v)})^{(x+v)^{-1}}$  and raises it to  $x$  with random number  $e$ , i.e.  $g^{y(x+e)}$ .
6. AMP is very simple in its structure so that it is easy to understand the protocol.
7. AMP is favorable to upgrading the existing system; AMP accommodates any kinds of salt schemes without degrading its performance so that the existing password file of various systems can be upgraded easily.

<sup>6</sup>Only AMP<sup>n</sup> is a pure password-based protocol. It is also comparable to the well-known related protocols such as EKE, SPEKE, SNAPI, PAK[7, 18, 28, 10] under the naked assumption such that the protocol is vulnerable to the password file compromise.

## 5 Conclusion

In this paper, we introduced a new verifier-based protocol, AMP, for secure password authentication and the Diffie-Hellman key agreement, by following the previous notable methods. In addition, we presented several variants of AMP. They are  $AMP^i$ ,  $AMP^n$ ,  $AMP^+$ ,  $AMP^{++}$ . Among them,  $AMP^n$  was a pure password-based protocol rather than a verifier-based protocol. Compared to the similarly secure protocols, AMP was the most efficient one. AMP holds the provable security, the best efficiency, the light constraints, and the generalization features as its advantages. In addition, it allows an easy integration to the existing systems.

Internet business and commercial services are growing rapidly while personal privacy and security concerns are slower than those activities. Authentication is undoubtedly very important. Though the hardware-dependent authentication methods are growing steadily, the pure password authentication scheme is still the most reasonable in a distributed environment, and the public-key based cryptographic protocol is the best solution for improving its security. We should note that the only password authentication method can truly authenticate the human mind over the network. Therefore, we might keep utilizing it over the Internet and in mobile environments even with the hardware-supported authentication schemes.

**Acknowledgment** Author of this document deeply appreciates Mr. David Jablon for his most helpful comments on AMP. More discussions and comments on AMP by IEEE P1363 Study Group would be greatly appreciated.

## References

- [1] R.Anderson and T.Lomas, "Fortifying key negotiation schemes with poorly chosen passwords," Electronics Letters, vol.30, no.13, pp.1040-1041, 1994
- [2] R.Anderson and S.Vaudenay, "Minding your  $p$ 's and  $q$ 's," Asiacrypt'96, LNCS, 1996
- [3] M.Bellare and P.Rogaway, "Entity authentication and key distribution," Crypto 93, LNCS 773, 1993
- [4] M.Bellare, R.Canetti, and H.Krawczyk, "A modular approach to the design and analysis of authentication and key exchange protocols," STOC 98, pp.419-428, 1998
- [5] M. Bellare, D. Pointcheval and P. Rogaway, "Authenticated key exchange secure against dictionary attack," To appear in Eurocrypt 2000
- [6] M.Bellare and P.Rogaway, "The AuthA protocol for password-based authenticated key exchange," Contribution to the IEEE P1363 study group for Future PKC Standards, available from <http://grouper.ieee.org/groups/1363/StudyGroup/submissions.html#autha>.

- [7] S.Bellovin and M.Merritt, "Encrypted key exchange : password-based protocols secure against dictionary attacks," Proc. IEEE Comp. Society Symp. on Research in Security and Privacy, pp. 72-84, 1992
- [8] S.Bellovin and M.Merritt, "Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password-file compromise," Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 244-250, 1993
- [9] M.Boyarsky, "Public-key cryptography and password protocols: the multi-user case," the 6th ACM Conference on Computer and Communication Security, September 16, 1999
- [10] V. Boyko, P. MacKenzie and S. Patel, "Provably secure password authenticated key exchange using Diffie-Hellman," To appear in Eurocrypt 2000
- [11] D.Denning, G.Sacco, "Timestamps in key distribution protocols," Commun. ACM, vol.24, no.8, pp.533-536, 1981
- [12] W.Diffie and M.Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol.22, no.6, pp.644-654, Nov. 1976
- [13] Y.Ding and P.Hoster, "Undetectable on-line password guessing attacks," ACM Operating Sys. Review, vol.29, no.4, pp.77-86, Oct. 1995
- [14] T.ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," IEEE Trans. Information Theory, vol.IT-31, no.4, pp.469-472, 1985
- [15] L.Gong, M.Lomas, R.Needham, and J.Saltzer, "Protecting poorly chosen secrets from guessing attacks," IEEE Journal on SAC., vol.11, no.5, pp.648-656, June 1993
- [16] L.Gong, "Optimal authentication protocols resistant to password guessing attacks," IEEE Comp. Security Foundation Workshop,, pp. 24-29 June 1995
- [17] S.Halevi and H.Krawczyk, "Public-key cryptography and password protocols," The 5th ACM Conference on Computer and Communications Security, 1998
- [18] D.Jablon, 'Strong password-only authenticated key exchange', ACM Comp. Comm. Review, vol.26, no.5, pp.5-26, 1996
- [19] D.Jablon, "Extended password key exchange protocols," WETICE Workshop on Enterprise Security, 1997
- [20] D.Jablon, Personal Communication, May 2000
- [21] T.Kwon and J.Song, "Efficient key exchange and authentication protocols protecting weak secrets," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E81-A, no.1, pp.156-163, January 1998

- [22] T.Kwon and J.Song, "Secure agreement scheme for  $g^{xy}$  via password authentication," Electronics Letters, vol.35, no.11, pp.892-893, 27th May 1999
- [23] T.Kwon, "Ultimate solution to authentication via memorable password," Manuscript
- [24] T.Kwon, "Avoiding information leakage in authentication protocol," Manuscript
- [25] C.Lim and P.Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup," Crypto 97, pp.249-263, 1997
- [26] M.Lomas, L.Gong, J.Saltzer, and R.Needham, "Reducing risks from poorly chosen keys," Proceedings of the 12th ACM Symposium on Operating System Principles, ACM Operating Systems Review, 1989, pp.14-18
- [27] S.Lucks, "Open key exchange: how to defeat dictionary attacks without encrypting public keys," The Security Protocol Workshop '97, April 7-9, 1997
- [28] P.MacKenzie and R.Swaminathan, "Secure network authentication with password identification," Presented to IEEE P1363a, August 1999
- [29] A.Menezes, P.van Oorschot, S.Vanstone, *Handbook of applied cryptography*, CRC Press, Inc., 1997
- [30] K.Nyberg and R.A.Rueppel, "Message recovery for signature scheme based on the discrete logarithm problem," Eurocrypt 94, pp. 182-193, 1994
- [31] P.van Oorschot and M.Wiener, "On Diffie-Hellman key agreement with short exponents," EUROCRYPT 96, pp. 332-343, 1996
- [32] S.Patel, "Number theoretic attacks on secure password schemes," IEEE Symposium on Security and Privacy, 1997
- [33] S.Pohlig and M.Hellman, "An improved algorithm for computing logarithms over  $GF(p)$  and its cryptographic significance," IEEE Transactions on Information Theory, vol.24, no.1, pp.106-110, 1978
- [34] J.Pollard, "Monte carlo methods for index computation mod  $p$ ," Mathematics of Computation, vol.32, pp.918-924, 1978
- [35] M.Roe, B.Christianson, D.Wheeler, "Secure sessions from weak secrets," Technical report from University of Cambridge and University of Hertfordshire, 1998
- [36] C.P.Schnorr, "Efficient identification and signatures for smart cards," Crypto 89, LNCS, pp.239-251, 1989
- [37] M.Steiner, G.Tsudik, and M.Waidner, "Refinement and extension of encrypted key exchange," ACM Operating Sys. Review, vol.29, no.3, 1995, pp.22-30

- [38] G.Tsudik, E.van Herreweghen, "Some remarks on protecting weak keys and poorly-chosen secrets from guessing attacks," Proc. 6th IEEE Comp. Security Foundation Workshop, 1993, pp.136-142
- [39] V.Voydoc and S.Kent, "Security mechanisms in high-level network protocols," Computing Surveys, vol.15, no.2, June 1983, pp.135-171
- [40] T.Wu, "Secure remote password protocol," Internet Society Symposium on Network and Distributed System Security, 1998

## Appendix : Genealogy of Password Protocol

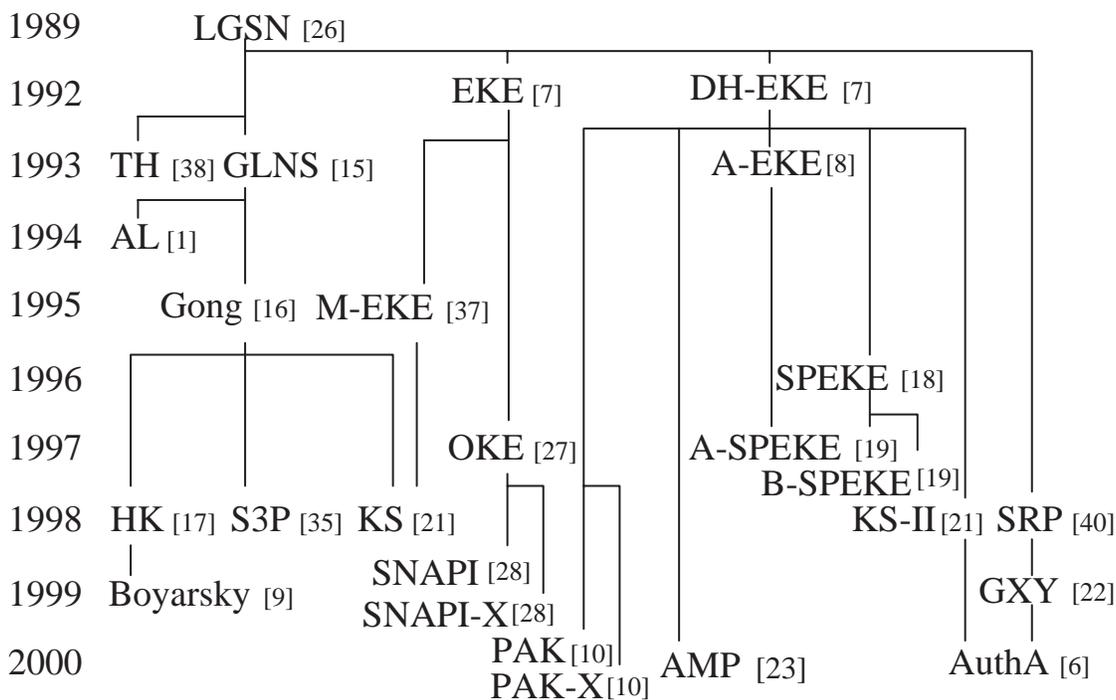


Figure 7. Password Authentication Protocols <sup>7</sup>

---

<sup>7</sup> The above genealogy is typically based on the opinion of the author. We analyzed all the protocols carefully and arranged them in the figure by considering their similarity or improvement. However, each author of the protocols could have different opinions. At this moment, we would like to make it clear that the above genealogy is only one of good references.