Authentication and Key Agreement via Memorable Password

Taekyoung Kwon*

July 27, 2000

Preliminary Version 2.2

Abstract

This paper presents our on-going work, "authentication and key agreement via memorable password." 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. The protocol nameed AMP is a new password authentication and key agreement protocol based on the amplified password proof idea. A party commits the high entropy information and amplifies her password with that information. She never shows even the amplified password for the proof, rather she shows the fact of knowing it. Our amplified password proof idea is very similar to the zero-knowledge proof in that sense. AMP mainly provides the password-verifier based authentication and the Diffie-Hellman based 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 such as AMP^i, AMP^n, AMP$^+$ and AMP^{++} are also proposed. Among them, AMP^n provides a pure password-based authentication. In the end, we give a comparison to the related protocols and discuss several promising applications on the Internet.

This manuscript is a preliminary version of our paper so that especially the proof story must be revised and improved. Any comment or suggestion would be always welcome. The protocols described in this paper were submitted as a contribution to the IEEE P1363 study group for future PKC standards, *Ultimate Solution to Authentication via Memorable Password* [23].

Keywords: Authentication, key agreement, password guessing, password verifier, public-key cryptography, discrete logarithm problem, Diffie-Hellman problem.

*Soda Hall, Computer Science Division, EECS, University of California, Berkeley, CA 94720, tkwon@cs.berkeley.edu or ktk@emerald.yonsei.ac.kr.

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.[1] Since the first scheme, LGSN[25], was introduced in 1989, many protocols have followed it. Among them, EKE[7] was a great 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[38] 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[37, 1, 36, 16, 21, 18, 19, 34, 39, 17]. However, some of them have been broken and some are still being cryptanalyzed[2, 13, 31, 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[26, 3] and was followed by SNAPI[27]. 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, 39, 22, 27, 6, 10]. The verifier-based protocols allow the asymmetric model in which a client possesses a password, while a server stores

[1] 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[39] 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[39]. 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[27]. AuthA was newly proposed but very similar to the previous protocols[6]. PAK-X was enhanced from PAK[10].

CONTRIBUTION. Our goal is to design a new protocol in a provable manner[2], which combines the following functions securely and efficiently.

- Password(-verifier) based authentication[8]
- Diffie-Hellman based key agreement[12]
- Easy generalization [28]

For this purpose, we propose a simple idea called the amplified password proof. In such a point of view, we name our protocol AMP that stands for Authentication and key agreement via Memorable Password. Regarding several functional issues, we also present four major variants of AMP. They are called AMP^i, AMP^n, AMP$^+$, AMP^{++}. Several minor variants also can be considered but they are explained implicitly. We compares the efficiency of all verifier-based protocols. Actually, AMP is the most efficient protocols among the existing verifier-based protocols. We also discuss several promising applications for those 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. π and τ denotes 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. ϕ 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. $\varphi() : \{0,1\}^* \rightarrow \{0,1\}^{>l(k)}$ means a one-way function that perturbs its image. All hash functions are assumed to behave like random oracles for security proof[3]. Note that we abbreviate a modular notation, mod p, for convenience hereafter.

RANDOM ORACLE. We assume random oracles $h_i() : \{0,1\}^* \rightarrow \{0,1\}^{l(k)}$ for $i \in [1,5]$. If *Eve* sends queries $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, ...$ to the random oracle h_i, she can receive answers $h_i(Q_j)$,

[2] Author of this paper understand the proof story must be revised and improved in the near future.

all independently random values, from the oracle. 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. Security of AMP is based on two familiar hard problems which are believed infeasible to solve in polynomial time. One is Discrete Logarithm Problem; given a prime p, a generator g of a multiplicative group Z_p^*, and an element $g^x \in Z_p^*$, find the integer $x \in [0, p-2]$. The other is Diffie-Hellman Problem; given a prime p, a generator g of a multiplicative group Z_p^*, and elements $g^x \in Z_p^*$ and $g^y \in Z_p^*$, find $g^{xy} \in Z_p^*$. These two problems hold their properties in a prime-order subgroup[30, 28]. 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 its main operation, a modular multiplication, for easy generalization. For the purpose, Bob chooses g that 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[30, 32, 33]. It is easy to make g by $\alpha^{(p-1)/q}$ where α 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 $p = 2q + 1$[24]. However, we strongly recommend to use a secure prime modulus p. Such a modulus should make our protocols secure and efficient.

2.2 Our Idea

The intrinsic problem of human-memorable password authentication is that the password itself has low entropy. Thus, our idea is to "amplify" such low entropy on the well-structured cryptographic protocol, i.e., on the secure interaction between communicating parties.

DEFINITIONS. Firstly we give some useful definitions and introduce the detailed idea.

Definition 1 *A Password Proof defines; a party A who knows a low entropy secret called a password makes a counterpart B convinced that A is who knows the password.*

There are two kinds of setup for the password proof(PP) such as a symmetric setup and an asymmetric setup. A client possesses a password while a server stores its verifier in the asymmetric setup model. This model benefits from salt for overcoming the text-equivalence of the symmetric setup. The password proof is actually composed of two kinds of proof.

Definition 2 *A Secure Password Proof defines; a party A successfully performs the Password Proof without revealing any information about the password itself.*

Definition 3 *An Insecure Password Proof defines; a party A successfully performs the Password Proof but fails the Secure Password Proof, or a party A successfully performs the Password Proof by showing some or all information about the password.*

The insecure Password Proof can be classified into the fully insecure password proof such as PAP, the partially insecure password proof such as CHAP, and the cryptographically insecure password proof such as some cryptographic protocols.

Definition 4 *An Amplified Password Proof defines; a party A who knows a password amplifies the password and makes a counterpart B convinced that A is who knows the amplified password.*

Theorem 1 *The Amplified Password Proof is a Secure Password Proof.*

AMP will be the protocol that enables the amplified password proof and the key agreement.

THE AMPLIFICATION. Our amplification idea is very simple that *Alice* insists on her knowledge of password π by giving $x + \pi$ rather than π only, while x is the randomly-chosen high entropy information. For the purpose, fresh x must be committed by *Alice* prior to her proof of each session. ($x + \pi$ is not guessable at all whereas π is guessable, if x is kept secret.)

Definition 5 *The Amplified Password ϕ defines $x + \pi \bmod q$ where x is chosen randomly at Z_q and π is a human-memorable password. (Note: ϕ is rather ephemeral.)*

We configure this idea as an amplified password proof.

THE AMPLIFIED PASSWORD PROOF. Assume that *Alice* knows π while *Bob* knows g^π. The procedure of the amplified password proof is composed of basically three steps such as initial commitment, challenge, and response; the initial commitment step performs a secure commitment of x having high entropy by *Alice*; the challenge step performs a random challenge of y by *Bob*; the response step performs a knowledge proof of *Alice* about the amplified password ϕ. We define three functions for each step; they are $\mathcal{G}_1()$ for initial commitment, $\mathcal{G}_2()$ for challenge, and $\mathcal{H}()$ for response.

Definition 6 *The Amplified Password Proof performs; Alice who knows her password π, randomly chooses and commits the high-entropy information x to Bob. Bob who knows g^π, picks y at random and asks Alice if she knows the password and the committed information. Alice responds with the fact she knows the amplified password ϕ.*

Alice		Bob
initial commitment	$\xrightarrow{\mathcal{G}_1(x)}$	
	$\xleftarrow{\mathcal{G}_2(y)}$	*challenge*
response	$\xrightarrow{\mathcal{H}(\phi)}$	

Due to the property of the secure commitment, $\mathcal{G}_1()$ should not reveal x even to *Bob*, for example, g^x. While $\mathcal{G}_2()$ transmits a fresh challenge, $\mathcal{H}()$ implies the fact only that *Alice* knows ϕ without revealing any information about x and π. If $\mathcal{G}_2() = g^{\phi y}$, then only who knows x and π can remove ϕ from $\mathcal{G}_2()$, i.e., $(g^{\phi y})^{\phi^{-1}} = g^y$. Undoubtedly *Bob* who knows $\mathcal{G}_1()$ as well as g^π, can make $\mathcal{G}_2()$ by $(g^x g^\pi)^y$ where $\phi = x + \pi$. As a result, both parties can make g^{xy} due to the Diffie-Hellman scheme so that we can set $\mathcal{H}() = g^{xy}$. This means *Alice* never shows the password itself for her proof, rather she proves the fact of knowing it. The amplified password proof idea is very similar to the zero-knowledge proof in that sense, but g^π must be kept secure because it is vulnerable to guessing attacks.

The Amplification and Key Exchange. It is easy to add key exchange to the amplified password proof because we already utilized the Diffie-Hellman scheme in describing the amplified password proof. In the amplified password proof, *Alice* showed g^{xy} for her proof of knowing ϕ. For key exchange instead, she can derive a session key from g^{xy} and show she agreed on it. *Bob* is also able to derive the same key and optionally show he also agreed on it. A strong one-way hash function must be the best tool for this capability. For mutual key confirmation as well as mutual authentication, however, the protocol must be configured by four steps rather than three steps to add *Bob*'s response. The mutual authenticatoin adds that *Alice* authenticates *Bob* who knows g^π. The following definition gives our full conceptual model.

Definition 7 *The conceptual model of* AMP *defines; Alice who knows π, says hello to Bob. Bob asks Alice if she knows π. Alice proves her knowledge of π and g^{xy}, and asks if he knows g^π and g^{xy}. Then, Bob proves his knowledge of g^π and g^{xy}. Finally, they agree on the same secret g^{xy} as well as authenticate each other.*

Alice		Bob
$x \in_R Z_q$, π		$y \in_R Z_q$, g^π
hello?	$\xrightarrow{\mathcal{G}_1(x)}$	
	$\xleftarrow{\mathcal{G}_2(y)}$	*hi, do you know* π?
yes, do you know g^π?	$\xrightarrow{\mathcal{H}_1(\phi)}$	
	$\xleftarrow{\mathcal{H}_2(\phi)}$	*Sure, I know it.*

Note that only *Alice* is able to know explicitly she has been authenticated by *Bob*. The amplifier password proof and key exchange must be the password authenticated key exchange.

2.3 Protocol Description

The followings are describing the setup and the run phases of our protocol, AMP. Basically AMP is designed as a verifier-based protocol. Therefore, we introduce verifier and salt in the asymmetric setup.

THE VERIFIER. Asymmetric setup is preferred because of the weaknesses of text-equivalence in symmetric setup[8, 19, 39]. Thus, we need a verifier $\mathcal{V}$ computed from π and salt τ. We define two verifiers; an explicit verifier ν known to a server and an implicit verifier υ obtained by a client. ν is easily computable from υ whereas the reverse is not easy to compute.

Definition 8 *Our password-verifiers are defined by,*

- implicit verifier : $\upsilon = \varphi(\tau, \pi)$ (similar to a private-key).
- explicit verifier : $\nu = g^{\upsilon}$ (similar to a public-key),

Two verifiers can be used in the way that a compromised password-file shows the explicit verifier whereas the implicit verifier is actually necessary for authentication. Both verifiers are vulnerable to the guessing attacks in the presence of τ so that they must be maintained securely. If *Eve* knows τ and ν, then she can find υ by the guessing attack or impersonate *Bob*; this is the inevitable feature of the verifier-based protocols. $\varphi()$ is a one-way function that merely perturbs and expands τ and π into a secure exponent[30]. Note that we replace π with υ and ν for asymmetric setup so that $\phi = x + \upsilon$ in our conceptual model.

SALT. We assume salt τ is disclosed in every protocol run. For example, τ 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 have implicit salt by making both parties get τ respectively without any exchange, e.g., from their identities[6, 10], though it is not favorable for upgrading the existing salt systems.

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[3].
3. *Bob* chooses $\tau \in_R \{0,1\}^{t(k)}$ and stores $(id, \tau, \nu = g^{\upsilon})$ where $\upsilon = h_1(\tau, \pi)$.
4. *id* indicates precisely a user name.

[3]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.

Bob should throw away π and υ but keep τ and ν.

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

Figure 1. AMP Protocol

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 τ and ν, 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 $\mathcal{G}_2 = (\mathcal{G}_1)^y (\nu)^y = (g^x \nu)^y = g^{\phi y}$ where $\phi = x + \upsilon$. He sends $(\tau, \mathcal{G}_2)$ to *Alice*.

3. After receiving message 2, *Alice* computes $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\upsilon = h_1(\tau, \pi)$, $\chi = (x + \upsilon)^{-1}(x + e) \bmod q$ and $\alpha = (\mathcal{G}_2)^\chi$. Note that $\alpha = (g^{\phi y})^{\phi^{-1}(x+e)} = g^{y(x+e)}$ where $\phi = x + \upsilon$. She computes $\mathcal{K}_1 = h_3(\alpha)$ and $\mathcal{H}_{11} = h_4(\mathcal{G}_1, \mathcal{K}_1)$. She sends *Bob* $\mathcal{H}_{11}$.

4. While waiting for message 3, *Bob* computes $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\beta = (\mathcal{G}_1)^y g^{ey} = (g^x g^e)^y = g^{(x+e)y}$, $\mathcal{K}_2 = h_3(\beta)$ 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, he computes $\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$ and sends *Alice* $\mathcal{H}_{22}$. This means he authenticated *Alice* who knows ϕ (actually υ and thus π since x is secure from g^x), 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 ν.

2.4 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 for withstanding *Eve* who stole ν, i.e., the password file compromise. For example, if *Eve* knowing ν sends ν^x to *Bob*, then *Bob* will reply with $(\nu^x\nu)^y$ and compute $\mathcal{K}_2 = (\nu^x g^e)^y = g^{(\upsilon x+e)y}$. However, *Eve* has only ν, x, e and $\nu^y (= \mathcal{G}_2^{(x+1)^{-1}})$ so that she still cannot find the key withoug knowing υ. 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 τ 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 υ such that $\upsilon = h_1(\tau, h(\tau, \pi))$ 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 hash functions of 128-bit hash are used, for example, MD5, we define $\varphi(x) = h(00|x|00|x)h(10|x|10|x)$ instead of $h_1(x)$.

3 AMP Protocol Proof

This section describes a proof story of AMP in the random oracle model.

3.1 A Communication Model

Firstly we formalize the communication model of our protocol on the basis of the work of Bellare and Rogaway[3]. That is, all communication among interacting parties is under the adversary's control.

THE PROTOCOL. The protocol can be formally specified by an efficiently computable function Π for two players; let us set $I \in \{A, B\}$ for *Alice* and *Bob*. Note that *Eve* is not included in the players[3]. Each party can be formally modeled by an infinite collection of oracles.

Definition 9 *Our protocol is a set of function* $\Pi_I(1^k, \sigma, \kappa, r) = (m, \delta, \mu)$ *for* I.

- 1^k : the security parameter, $k \in \mathcal{N}$.

- $\sigma \in \{0,1\}^*$: the secret information of the sender.
- $\kappa \in \{0,1\}^*$: the conversation so far.
- $r \in \{0,1\}^\infty$: the random coin flips of the sender.
- $m \in \{0,1\}^* \cup \{*\}$: the next message for a reply.
- $\delta \in \{Accept, Reject, *\}$: the decision.
- $\mu \in \{0,1\}^* \cup \{*\}$: the agreed session key.

The value $\{*\}$ means; (1) there is no reply message for m, (2) the decision is not yet made for δ, (3) the decision is neither yet made nor accepted for μ. $\mathcal{N}$ means a set of integer. For example, Π_A^s indicate that *Alice* computes on the message from *Bob* and gives out an output in session s. That is, each party is formally modeled by an infinite collection of oracles; Π_A^i and Π_B^j where i and $j \in \mathcal{N}$ indicate the instances. Therefore, *Eve* is able to call the oracles, Π_A^i and Π_B^j, and attempts to obtain desired information.

THE LONG-LIVED WEAK-KEY GENERATOR. A long-lived weak-key(LW-key) generator is $\mathcal{W}(1^{\omega(k)}, \iota, r_G)$ where $\iota \in I \cup \{E\}$ and $r_G \in \{0,1,\}^\infty$. Note that the LW-key may have a length of k but its entropy is totally different[4]. When we assume the strong-key length is only k, i.e., our security parameter, the parameter $\omega(k)$ means the low entropy of the LW-key. Brute-forcing $2^{\omega(k)}$ values is feasible whereas k is large enough against brute-forcing 2^k values (hence $2^{\omega(k)} << 2^k$). The point of the LW-key is that the adversary is denied by the generator as like the long-lived key case[3] but it is acceptable in the probability of $2^{-\omega(k)}$. Our model agrees on $\mathcal{W}(1^{\omega(k)}, A, r_G) = \pi$, $\mathcal{W}(1^{\omega(k)}, B, r_G) = \nu$, and $\mathcal{W}(1^{\omega(k)}, E, r_G) = \lambda$.

THE ADVERSARY. The adversary *Eve* is represented as a probabilistic machine $E(1^k, \sigma_E, r_E)$ equipped with an infinite collection of oracles Π_I^s for $i \in I$ and $s \in \mathcal{N}$[3].

Let $Pr[] \leq 2^{-k}$ be a negligible probability for our security parameter k. *Eve* is allowed to do everything she wants except for solving the discrete logarithm problem as well as the Diffie-Hellman problem, and finding out a hidden-value in a negligible probability. Therefore, we can say; $\{Pr[\text{Discrete-Log}^E(k)], Pr[\text{Diffie-Hellman}^E(k)]\} < 2^{-k}$ where $\text{Discrete-Log}^E(k)$ and $\text{Diffie-Hellman}^E(k)$ are such events.

When the adversary is deterministic and restricts its action to faithfully conveying each flow among oracles, i.e., matching conversations, she is called a benign adversary[3]. Let $\text{No-Matching}^E(k)$ be the event that Π_i^s is accepted and there is no oracle Π_j^t which engaged in a matching conversation.

[4] Actually the LW-key, i.e., the password, is chosen by a human-user through a limited input device such as a keyboard or keypad, and in a small set of human-memorable word space.

Eve communicates with the oracles via queries of the form $\mathcal{Q}(i, s, n)$; *Eve* sends message n to the oracle of i. There are some special queries for adversary such as $\mathcal{Q}(i, s, guess)$ for searching at most $2^{\omega(k)}$ space with the LW-key generator, and $\mathcal{Q}(i, s, compromise)$ for compromising a verifier. Note that a compromise query converts s to a compromised session for handling a password file compromise[8]. Also note that the number of failures of the query $\mathcal{Q}(i, s, guess)$ asked to fresh oracles is counted globally. We define that counter $\mathcal{C}_i$ for $i \in I$. If Π_i^s has accepted, *Eve* is able to send other special queries; $\mathcal{Q}(i, s, reveal)$ for compromising a session key, $\mathcal{Q}(i, s, corrupt)$ for compromising a password and $\mathcal{Q}(i, s, test)$ for measuring adversarial success. Note that a corrupt query converts s to a corrupted session for handling perfect forward secrecy[18], while a reveal query converts s to a revealed session for handling a known-key attack (Denning-Sacco attack)[11].

THE SESSIONS. Depending on the ability of the adversary, we can classify considerable sessions as follows. There must be FreshSession and UnfreshSession. They can be converted to SucceededSession or FailedSession. SucceededSession can be devided into MatchedSession and No-matchedSession. Each session could allow the adversary to have some valid information before running or examining those sessions. FreshSession can be devided into PureFreshSession in which valid information is never provided, and CompromisedButFreshSession where the verifier ν is provided. UnfreshSession can be devided into RevealedSession that provides $g^{(x+e)y}$ or $\mathcal{K}$, and and CorruptedSession provides π. Note that CompromisedUnfreshSession is negligible for examination because off-line guessing attacks on ν is inevitable in every protocol. The adversary is allowed to use all of these session for achieving her goals. The query will be answered by Π_i^s by the following experiment.

RUNNING THE PROTOCOL. Running a protocol Π (with the LW-key generator $\mathcal{W}$) in the presence of *Eve* and k, means performing the following experiment in a given session:

1. Choose a string $r_G \in_R \{0,1\}^\infty$ and $\sigma_i = \mathcal{W}(1^k, i, r_G)$, for $i \in I$, and set $\sigma_E = \mathcal{W}(1^k, E, r_G)$.
2. Choose a string $r_i^s \in_R \{0,1\}^\infty$ for $i \in I$ and $s \in \mathcal{N}$, and a string $r_E \in_R \{0,1\}^\infty$.
3. Let $\kappa_i^s = \lambda$ for all $i \in I$ and $s \in \mathcal{N}$.
4. Run the adversary, $E(1^k, \sigma_E, r_E)$, answering oracle calls as follows. When E asks a query, $\mathcal{Q}(i, s, n)$, oracle Π_i^s answers with (m, δ) by computing $(m, \delta, \mu) = \Pi_I(1^k, \sigma, \kappa_i^s.n, r_i^s)$, and sets $\kappa_i^s = \kappa_i^s.n$.
5. The adversary chooses an oracle Π_i^s and attempts to guess its session key or password.

While running the protocol, the adversary asks all queries she wishes to ask.

SECURITY DEFINITION. The following definition is derived from work of Bellare and Rogaway[3], and the following work of Stefan Lucks[26].

Definition 10 *Protocol* Π *is a* **secure authenticated key exchange with a LW-key generator** $\mathcal{W}()$ *if the following statements are true:*

1. If two oracles have matching conversations, then both oracles accept and agree on the identical key.
2. If it is the case that the oracles accept and agree on the same key, then the probability of no-matching is negligible.
3. If *Eve* is benign, her probability of success is negligible.
4. If *Eve* has been rejected R times, the possible set of v decreases linearly, $2^{\omega(k)} - R$.
5. If *Eve* has been rejected $R(< 2^{\omega(k)} - 1)$ times but finally remains benign, her probability of success is still negligible.

The above security definition must be improved and refined in the near future. The first condition considers the completeness of the protocol. The second condition implies that adversaries cannot be accepted by the oracle without knowing the valid password (or the valid verifier) and the valid key. The third condition regards the security against the passive adversary who always listens to the conversation and analyzes the eavesdropped message. The fourth condition holds that an on-line trial cannot partition out the searching space. The fifth condition deals with security against the active adversary who occasionally participates in the conversation but mostly analyzes the gathered information off-line.

3.2 Security Examination

We show the security of our protocol by inducing that the probability of success for adversary is negligible. Our most favorite tools are, of course, Discrete-Log$^E(k)$ and Diffie-Hellman$^E(k)$.

Lemma 1 *The probability of success is negligible for forging* $\mathcal{G}_1$ *or* $\mathcal{G}_2$ *in* FreshSession *of* AMP.

Proof Sketch: The adversary *Eve* is allowed to ask $\mathcal{Q}(i, s, \mathcal{G}_1)$ or $\mathcal{Q}(i, s, \mathcal{G}_2)$ for FreshSession.

1. Let *Eve* choose $x \in_R Z_q$ and ask $\mathcal{Q}(B, s, g^x)$ in PureFreshSession. Then Π_B responds with $\mathcal{G}_2 = g^{(x+v)y}$. *Eve* could find $\alpha' = g^{y'(x+e)}$ by computing ${\mathcal{G}_2}^{(x+v')^{-1}(x+e)}$ and asking $\mathcal{Q}(B, s, guess)$ with $v' \in_R \{0,1\}^{\omega(k)}$. However, she cannot verify $\alpha' \doteq \beta$, i.e., $g^{y'(x+e)} \doteq g^{(x+e)y}$, without submitting $\mathcal{H}_1'$ ahead of $\mathcal{H}_2$. The probability of successful submission is $2^{-\omega(k)}$. $\mathcal{C}_B$ must count up the number of failures so that her attack can be detected easily in only R trials where R is very small such that $R << 2^{\omega(k)/2}$.

2. Let *Eve* choose $x \in_R Z_q$ and ask $\mathcal{Q}(B, s, \nu^x)$ in CompromisedButFreshSession. Note that guessing attacks on ν is not a concern in CompromisedButFreshSession. Then Π_B responds with $\mathcal{G}_2 = g^{(x+1)\upsilon y}$. *Eve* could find $g^{\upsilon y}$ simply by $\mathcal{G}_2^{(x+1)^{-1}}$. However, as long as she is not given υ, she cannot compose $g^{(x+e)y}$ without finding y. Finding y is bounded by $Pr[\text{Discrete-Log}^E(k)]$ so that it is negligible.

3. Let *Eve* choose $c \in_R Z_q$ and ask $\mathcal{Q}(A, s, g^c)$ where she is given $\mathcal{G}_1$ in PureFreshSession. Then Π_A responds with $\mathcal{H}_1$ by getting $\alpha = (\mathcal{G}_2^{'})^{(x+\upsilon)^{-1}(x+e)}$. We can rewrite $c = (x + \upsilon^{'})y^{'} \bmod q$ where $\upsilon^{'}$ and $y^{'}$ are variables. *Eve* must find $y^{'}$ such that $y^{'} \equiv c(x+\upsilon^{'})^{-1} (\bmod\ q)$ for getting $\beta^{'} = (\mathcal{G}_1 g^e)^{y^{'}}$ and verifying $\alpha \doteq \beta^{'}$. For the computation of $y^{'}$, it is necessary to know x of $\mathcal{G}_1$. However, getting x from $\mathcal{G}_1$ is bounded by $Pr[\text{Discrete-Log}^E(k)]$ so that it is negligible.

4. Let *Eve* choose $y \in_R Z_q$ and $\upsilon^{'} \in_R \{0,1\}^{\omega(k)}$, and ask $\mathcal{Q}(A, s, \mathcal{G}_2^{'})$ where she is given $\mathcal{G}_1$ in PureFreshSession and $\mathcal{G}_2^{'} = (\mathcal{G}_1 g^{\upsilon^{'}})^y$. Then Π_A responds with $\mathcal{H}_1$ by computing $\alpha = (\mathcal{G}_2^{'})^{(x+\upsilon)^{-1}(x+e)}$, but note that $\alpha = (g^{(x+\upsilon')y})^{(x+\upsilon)^{-1}(x+e)}$, i.e., $\alpha = g^{y'(x+e)}$ rather than $g^{y(x+e)}$ where $y^{'} \equiv (x+\upsilon')y(x+\upsilon)^{-1} (\bmod\ q)$. Finding $y^{'}$ or υ is the only way for *Eve* to be accepted by the oracle.

   (a) x chosen by Π_A, $x \in_R \{0,1\}^{>l(k)}$, is not given to *Eve*. We can say that $\upsilon = \upsilon^{'}$ if and only if $y^{'} \equiv y (\bmod\ q)$. It corresponds to the on-line attack.

      i. Let *Eve* attempt to verify $\upsilon \doteq \upsilon^{'}$. However, $\upsilon^{'}$ is rather a constant because she defined it before receiving $\mathcal{H}_1$ from Π_A. *Eve* cannot replace $\upsilon^{'}$ for further verification without retrying it on line. The probability of $\upsilon = \upsilon^{'}$ is $2^{-\omega(k)}$; an extremely low probability for on-line success. Due to the maximum count of on-line failure, $\mathcal{C}_A = R$, she must be denied by the oracle before trying $2^{\omega(k)} - R$ more guesses. The probability of $\upsilon \neq \upsilon^{'}$ is very high such that $Pr[] \leq 1 - \frac{1}{2^{\omega(k)} - R}$. Therefore, we can say hereafter $\upsilon^{'}$ is a constant such that $\upsilon \neq \upsilon^{'}$ in this case.

      ii. Let *Eve* attempt to find $y^{'}$ but she has to know x for attempting the equation, $y^{'} \equiv (x+\upsilon')y(x+\upsilon)^{-1} (\bmod\ q)$. Finding x from $\mathcal{G}_1$ is bounded by $Pr[\text{Discrete-Log}^E(k)]$ so that it is negligible. Even if *Eve* computes $\alpha^{'} = (\mathcal{G}_1 g^e)^y = g^{(x+e)y}$, it is clear that $\alpha \neq \alpha^{'}$ when $\upsilon \neq \upsilon^{'}$. Thus, the probability of finding $y^{'}$ is $Pr[] < 2^{-k}$.

   (b) Let *Eve* guess $\upsilon^{''} \in_R \{0,1\}^{\omega(k)}$ and compute $\alpha^{''} = \mathcal{G}_2^{''} g^{-\upsilon'' y} = g^{(x+e+\upsilon')y - \upsilon'' y}$ for verifying $\alpha^{''} \doteq \alpha$ in $2^{-\omega(k)}$ probability, rather than attempt to replace $\upsilon^{'}$ with $\upsilon^{''}$, where $\mathcal{G}_2^{''} = (\mathcal{G}_1 g^e g^{\upsilon^{'}})^y$. If $\alpha^{''} = \alpha$ with guessed $\upsilon^{''}$, she can be convinced $\upsilon = \upsilon^{''}$. Thus, if the equation,
   $(x + \upsilon^{'})y(x + \upsilon)^{-1}(x + e) \equiv (x + e)y + \upsilon^{'} y - \upsilon^{''} y (\bmod\ q)$

is true, then she can find v in $2^{-\omega(k)}$ probability, regardless of x and $v^{'}$. Otherwise, she has to find x first. We can rewrite it as,

$(x+v^{'})\not{y}(x+v)^{-1}(\not{x}+\not{e})\equiv(\not{x}+\not{e})\not{y}(1+v^{'}x^{-1}-v^{''}x^{-1})(\text{mod } q)$.

That is,

$(x+v^{'})(x+v)^{-1}\equiv(1+v^{'}x^{-1}-v^{''}x^{-1})(\text{mod } q)$.

We can transpose $(x+v)^{-1}$ so that,

$$\begin{aligned}(x+v^{'}) &\equiv (x+v)+(x+v)v^{'}x^{-1}-(x+v)v^{''}x^{-1}\\ &\equiv x+v+v'+vv^{'}x^{-1}-v^{''}-vv^{''}x^{-1}\\ &\equiv (x+v^{'})+(v-v^{''})+(v^{'}-v^{''})vx^{-1} \quad (\text{mod } q).\end{aligned}$$

Then, we can transpose $(x+v^{'})$ so that;

$0\equiv(v-v^{''})+(v^{'}-v^{''})vx^{-1}(\text{mod } q)$.

Therefore, the equality such that,

$v=v^{'}=v^{''}$ (due to $v=v^{''}$ and $v^{'}=v^{''}$),

is the mandatory requiremet of this modular equation. However, the probability of success is negligible because $v\neq v^{'}$ with very high probability as we mentioned above in (a). Therefore, *Eve* must find x for getting v. The probability of verifying $\alpha^{''}\doteq\alpha$ is $Pr[]<2^{-k}$.

After all, the probability of success is negligible for forged queries in FreshSession, because on-line retrial is detectable while off-line verification has a probability such that $Pr[]<2^{-k}$. □

The following theorem shows AMP is a secure authenticated key exchange protocol with a LW-key generator $\mathcal{W}()$ by following Definition 10. Security against conventional attacks will be examined in section 5.

Theorem 2 AMP *is a secure authenticated key exchange protocol with a LW-key generator* $\mathcal{W}()$.

Proof Sketch: We deal with each condition of Definition 10.

1. A completeness of the protocol in MatchedSession is already shown in Figure 1. If two oracles have matching conversations, then $\mathcal{H}_1$ and $\mathcal{H}_2$ can be verified successfully due to the fact that $\alpha=\beta(\mathcal{K}_1=\mathcal{K}_2)$.

2. The final acceptance means both $\mathcal{H}_1$ and $\mathcal{H}_2$ are successfully verified. Therefore, we have to scrutinize whether it is possible in No-matchedSession. If it is not true, we may have $Pr[\mathsf{No{-}Matching}^{E}(k)]\leq 2^{-k}$.

   (a) The birthday paradox is negligible due to the nature of the random oracle, $h_i():\{0,1\}^{*}\rightarrow\{0,1\}^{l(k)}$; the probability is $2^{-\frac{1}{2}l(k)}\leq 2^{-k}$.

(b) Due to item (a), the correct value $\mathcal{K}(=\mathcal{K}_1=\mathcal{K}_2)$ is mandatory for the acceptance even in sf No-matchedSession. For finding $\mathcal{K}$, *Eve* must obtain α or β due to the one-way property of random oracles.

i. The probability of guessing $g^{(x+e)y}(=\alpha=\beta)$ in PureFreshSession is $Pr[] < 2^{-k}$.

ii. Rewrite α and β where $\mathcal{G}_1 = g^{\log \mathcal{G}_1}$ and $\mathcal{G}_2 = (\mathcal{G}_1\nu)^{(\log \mathcal{G}_1+v)^{-1}\log \mathcal{G}_2}$, i.e., $\alpha = (\mathcal{G}_2)^{(\log \mathcal{G}_1+v)^{-1}(\log \mathcal{G}_1+e)} = (\mathcal{G}_1 g^e)^{(\log \mathcal{G}_1+v)^{-1}\log \mathcal{G}_2} = \beta$. For $\mathcal{G}_1$ there is nothing ahead, but for $\mathcal{G}_2$ we need $\mathcal{G}_1$. Note that $(\mathcal{G}_1, \mathcal{G}_2) \rightarrow e$. Thus, we can find easily the flows, $(\mathcal{G}_1 \rightarrow \mathcal{G}_2 \rightarrow e \rightarrow \alpha)$ and $(\mathcal{G}_1 \rightarrow \mathcal{G}_2 \rightarrow e \rightarrow \beta)$. Without such flows, the exponents of $\mathcal{G}_1$ and $\mathcal{G}_2$ must be analyzed but the probability is $Pr[] \leq 2^{-l(k)}$ even with a forged attempt by Lemma 1.

After all, we have $Pr[\mathsf{No\text{-}Matching}^E(k)] \leq 2^{-k}$ so that it is infeasible in No-matchedSession.

3. When *Eve* is benign, all she receives from the oracle are $\{id, \tau, \mathcal{G}_1, \mathcal{G}_2, \mathcal{H}_1, \mathcal{H}_2\}$ for every session where their internal values, x and y, are independent random values from $\{0,1\}^{>l(k)}$. Therefore, g^x, g^y, and their composition on the cyclic group must be well distributed on the group. We assume the uniform distribution. Since $\mathcal{W}(E) = \lambda$ and $\mathcal{G}_2 = (\mathcal{G}_1 g^v)^y$, the probability of finding $g^{y'}$ by guessing v' in $\mathcal{G}_1$ and $\mathcal{G}_2$, is less than $2^{-(\omega(k)+l(k))}$. For verifying the guess of v', *Eve* must find α or β for asking the random oracle. However, finding $g^{(x+e)y}$ over g^x and $g^{(x+v)y}$ without v must be bounded by $Pr[\mathsf{Diffie\text{-}Hellman}^E(k)]$ so that it is negligible. Therefore, the probability of success for benign *Eve* is $Pr[] < 2^{-k}$.

4. Since $\mathcal{G}_1$ and $\mathcal{G}_2$ remain on the cyclic group under uniform distribution, there is no way to find the relationship between the rejected guesses and the remaining guesses. Other possibilities are all negligible by Lemma 1. If *Eve* is rejected, she must reduce the set by one, $2^{\omega(k)} - 1$, and try again with another guess. That is, the set is reduced linearly. Therefore, the success probability of her on-line guess is only $Pr[] \leq \frac{1}{2^{\omega(k)}-R-c}$ for very small $c(\geq 0)$. She must be denied by the oracle only in R trials by Lemma 1.

5. Assume all $\mathcal{C}_i$s are set off and *Eve* has been rejected with different guesses $2^{\omega(k)} - 2$ times by the oracle, then she could have bernoulli trial on two remaining guesses; if one is rejected then the other is the one and vice versa. However, assume *Eve* does not participate in FreshSession any more but she only be benign in FreshSession. Then, she is only able to analyze all rejected messages and new eavesdropped messages equipped with bernoulli trial on guess. For actual participation, the probability was less than 2^{-k} by Lemma 1. For her analysis, the probability is $2^{-(l(k)+1)}(< 2^{-k})$ by item 3 of this proof. Hence, the probability of success for partially benign *Eve* is negligible. That means if *Eve* attempts off-line analysis even with a small dictionary, she does not have any advantage without knowing x or y for each message.

AMP is a secure authenticated key exchange protocol with a LW-key generator $\mathcal{W}()$. □

Lemma 2 *The adversary does not benefit from* RevealedSession *or* CorruptedSession *for achieving each goal.*

Proof Sketch: *Eve* attempts to find π or υ in RevealedSession while she attempts to find $\mathcal{K}$ or $\alpha(=\beta)$ in CorruptedSession. If *Eve* benefits from each session, the given information must make non-negligible probability of success or make some advantage for the query $\mathcal{Q}(i, s, test)$.

1. Let *Eve* ask $\mathcal{Q}(i, s, reveal)$. Then she is given $\mathcal{K}$ and $\alpha(=\beta)$ in RevealedSession. Due to the one-way property of random oracles, we assume $\alpha(=\beta)$ is given. Since $\alpha(=\beta)$ is not re-usable due to x and y, she cannot attempt to be granted on-line. For tracking to π or υ, she must be also in MatchedSession. Then we say she is given $\{id, \tau, e, g^x, g^{(x+\upsilon)y}, g^{(x+e)y}, \mathcal{H}_1, \mathcal{H}_2\}$ with matching-conversations. For verifying π' and υ', she should make $g^{(x+\upsilon')y}$ on the given information but she cannot make it without finding y. Otherwise, she has to find x for finding g^y from $g^{(x+e)y}$ and making $g^{y(x+\upsilon')}$. Both are still bounded by $Pr[\text{Discrete-Log}^E(k)]$. It is not difficult to understand RevealedSession is not advantageous to *Eve*.

2. Let *Eve* ask $\mathcal{Q}(i, s, corrupt)$. Then she is given π and υ in CorruptedSession. Due to the one-way property of random oracles, we assume π is given. For tracking to $\mathcal{K}$ or $\alpha(=\beta)$, she must be also in MatchedSession. Then she is also given $\{id, \tau, \pi, e, g^x, g^{(x+\upsilon)y}, \mathcal{H}_1, \mathcal{H}_2\}$ with matching-conversations. For making $g^{(x+e)y}$, she should remove ϕ from $g^{\phi y}$ and find y where $\phi = x + \upsilon$. Finding ϕ includes x so that both findings must be bounded by $Pr[\text{Discrete-Log}^E(k)]$. Even if we assume ϕ is removed, the problem is still bounded by $Pr[\text{Diffie-Hellman}^E(k)]$. It is not difficult to understand CorruptedSession is not advantageous to *Eve*. □

4 AMP Protocol Variants

We present four explicit variants of AMP. They are called AMP^i, AMP^n, AMP$^+$, AMP^{++}. AMP^i is extended for accommodating implicit salt while AMP^n is extended for non-verifier authentication similar to clear-text password authentication. AMP$^+$ and AMP^{++} are extended for disregarding the information leakage question though such are not critical in AMP.

4.1 AMP^i

AMP^i is a simple extension of AMP for accommodating implicit salt.

Protocol Setup. This step determines and publishes global parameters of 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 $(id, \nu = g^{\upsilon})$ where $\upsilon = h_1(id, B, \pi)$.

4. *id* indicates precisely a user name.

Bob should throw away π and υ but keep ν.

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

Figure 2. 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 $x \in_R Z_q$ and sends $(id, \mathcal{G}_1)$ to *Bob*.

2. After receiving message 1, *Bob* loads ν, 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 $\mathcal{G}_2 = (\mathcal{G}_1)^y (\nu)^y = (g^x \nu)^y = g^{\phi y}$ where $\phi = x + \upsilon$. He sends $\mathcal{G}_2$ to *Alice*.

3. While waiting for message 2, *Alice* computes $\upsilon = h_1(id, B, \pi)$ and $\xi = (x+\upsilon)^{-1} \bmod q$, i.e., $\xi = \phi^{-1}$. 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 $\alpha = (g^{\phi y})^{\phi^{-1}(x+e)} = g^{y(x+e)}$ where $\phi = x + \upsilon$. 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 $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\beta = (\mathcal{G}_1)^y g^{ey} = (g^x g^e)^y = g^{(x+e)y}$, $\mathcal{K}_2 = h_3(\beta)$ 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 ϕ (actually υ and thus π since x is secure from g^x), 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 ν.

AMP^i 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.

4.2 AMP^n

AMP^n 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^n is the most efficient but more sensitive compared to other AMPs (very similarly to comparing EKE with A-EKE). So we call AMP^n "AMP-naked" because it is not protected by e and salted ν.

Protocol Setup. This step determines and publishes global parameters of AMP^n.

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 $(id, \nu = g^{\upsilon})$ where $\upsilon = h_1(\pi)$; he can store υ rather than ν.
4. *id* indicates precisely a user name.

Bob should throw away π and υ but keep ν; note again that he can keep υ instead of ν.

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
input (id, π)		*store* (id, ν)
$x \in_R Z_q$		
$\mathcal{G}_1 = g^x$	$\xrightarrow{id, g^x}$	

		$fetch\ (id, \nu)$
$\upsilon = h_1(\pi)$		$y \in_R Z_q$
$\chi = (x + \upsilon)^{-1} x \bmod q$	$\xleftarrow{(g^x \nu)^y}$	$\mathcal{G}_2 = (\mathcal{G}_1 \nu)^y$
$\alpha = (\mathcal{G}_2)^\chi$		$\beta = (\mathcal{G}_1)^y$
$\mathcal{K}_1 = h_2(\alpha)$		$\mathcal{K}_2 = h_2(\beta)$
$\mathcal{H}_{11} = h_3(\mathcal{G}_1, \mathcal{K}_1)$	$\xrightarrow{h(g^x, \mathcal{K})}$	$\mathcal{H}_{12} = h_3(\mathcal{G}_1, \mathcal{K}_2)$
		$verify\ \mathcal{H}_{11} \doteq \mathcal{H}_{12}$
$\mathcal{H}_{21} = h_4(\mathcal{G}_2, \mathcal{K}_1)$	$\xleftarrow{h((g^x \nu)^y, \mathcal{K})}$	$\mathcal{H}_{22} = h_4(\mathcal{G}_2, \mathcal{K}_2)$
$verify\ \mathcal{H}_{21} \doteq \mathcal{H}_{22}$		

Figure 3. AMP^n 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 ν, 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. If *Bob* stores υ rather than ν for spatial efficiency, he computes $\mathcal{G}_2 = (\mathcal{G}_1)^y g^{\upsilon y}$. Note that $\mathcal{G}_2 = (\mathcal{G}_1)^y (\nu)^y = (g^x \nu)^y = g^{\phi y}$ where $\phi = x + \upsilon$. He sends $\mathcal{G}_2$ to *Alice*.

3. While waiting for message 2, *Alice* computes $\upsilon = h_1(\pi)$ and $\chi = (x + \upsilon)^{-1} x \bmod q$. After receiving message 2, *Alice* computes $\alpha = (\mathcal{G}_2)^\chi$. Note that $\alpha = (g^{\phi y})^{\phi^{-1} x} = g^{yx}$ where $\phi = x + \upsilon$. She computes $\mathcal{K}_1 = h_2(\alpha)$ and $\mathcal{H}_{11} = h_3(\mathcal{G}_1, \mathcal{K}_1)$. She sends *Bob* $\mathcal{H}_{11}$.

4. While waiting for message 3, *Bob* computes $\beta = (\mathcal{G}_1)^y = (g^x)^y = g^{xy}$, $\mathcal{K}_2 = h_2(\beta)$ 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 ϕ (actually either of ν, υ or π is acceptable, see below), 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 ν or υ.

AMP^n, 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 ν sends

ν^x to *Bob*, she can complete the protocol because $\mathcal{G}_2 = \nu^{(x+1)y}$ while $\beta = \nu^{xy}$[20]. However, AMP^n provides much more well-defined form of authentication for having EKE-security.

4.3 AMP$^+$

AMP$^+$ is a simple extension of AMP for disregarding the information leakage question such that "message and key are clearly unrelated?" Note that $\mathcal{G}_2 = g^{(x+v)y}$ and $\alpha = \beta = g^{(x+e)y}$; they are unrelated because random e is dependent on x and y but x and y are independent to each other. Protocol setup of AMP$^+$ 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.

(id, π) Alice		Bob (id, τ, ν)
$x \in_R Z_q$		
$\mathcal{G}_1 = g^x$	$\xrightarrow{id, g^x}$	
		$fetch\ (id, \tau, \nu)$
		$y \in_R Z_q$
$e_1 = h_2(\mathcal{G}_1, id, Alice, Bob)$		$e_1 = h_2(\mathcal{G}_1, id, Alice, Bob)$
	$\xleftarrow{\tau, (g^{xe_1}\nu)^y}$	$\mathcal{G}_2 = (\mathcal{G}_1^{e_1}\nu)^y$
$e_2 = h_3(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$		$e_2 = h_3(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$
$v = h_1(\tau, \pi)$		
$\chi = (xe_1 + v)^{-1}(x + e_2) \bmod q$		
$\alpha = (\mathcal{G}_2)^\chi$		$\beta = (\mathcal{G}_1 g^{e_2})^y$
$\mathcal{K}_1 = h_4(\alpha)$		$\mathcal{K}_2 = h_4(\beta)$
$\mathcal{H}_{11} = h_5(\mathcal{G}_1, \mathcal{K}_1)$	$\xrightarrow{h(g^x, \mathcal{K})}$	$\mathcal{H}_{12} = h_5(\mathcal{G}_1, \mathcal{K}_2)$
		$verify\ \mathcal{H}_{11} \doteq \mathcal{H}_{12}$
$\mathcal{H}_{21} = h_6(\mathcal{G}_2, \mathcal{K}_1)$	$\xleftarrow{h((g^x\nu)^y, \mathcal{K})}$	$\mathcal{H}_{22} = h_6(\mathcal{G}_2, \mathcal{K}_2)$
$verify\ \mathcal{H}_{21} \doteq \mathcal{H}_{22}$		

Figure 4. AMP$^+$ Protocol

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

1. *Alice* computes $\mathcal{G}_1 = g^x$ by choosing $x \in_R Z_q$ and sends $(id, \mathcal{G}_1)$ to *Bob*.
2. After receiving message 1, *Bob* loads τ and ν, and computes $e_1 = h_2(\mathcal{G}_1, id, Alice, Bob)$, $\mathcal{G}_2 = (\mathcal{G}_1^{e_1}\nu)^y$ by choosing $y \in_R Z_q$. This can be done by the simultaneous exponentiation method. Note that $\mathcal{G}_2 = (\mathcal{G}_1)^{e_1 y}(\nu)^y = g^{(xe_1+v)y}$. 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 $e_2 = h_3(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\upsilon = h_1(\tau, \pi)$, $\chi = (xe_1 + \upsilon)^{-1}(x + e_2) \bmod q$ and $\alpha = (\mathcal{G}_2)^{\chi}$. Note that $\alpha = (g^{(xe_1+\upsilon)y})^{(xe_1+\upsilon)^{-1}(x+e_2)} = g^{y(x+e_2)}$. 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 $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\beta = (\mathcal{G}_1)^y g^{e_2 y} = (g^x g^{e_2})^y = g^{(x+e_2)y}$, $\mathcal{K}_2 = h_4(\beta)$ 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 υ (actually, π), 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 ν.

We should define $h_6(x) = h(10|x|01)$ for AMP$^+$. The randomness of e_1 is totally dependent upon the randomness of $\mathcal{G}_1$ 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 $\mathcal{G}_1$. Note that $\mathcal{G}_2 = g^{(xe_1+\upsilon)y}$ while the agreed key is $g^{(x+e_2)y}$. It is clearer to see they are unrelated regarding the intractability of the discrete logarithm problem and the Diffie-Hellman problem.

4.4 AMP^{++}

AMP$^+$ is another form of extension for disregarding the information leakage question.

Protocol Setup. This step determines and publishes global parameters of 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 $(id, \nu = g^{-\upsilon})$ where $\upsilon = h_1(id, Bob, \pi)$[5].

Bob should throw away π and υ but keep *id* and ν.

Protocol Run. The following describes how to run AMP^{++}. 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.

Alice	Bob
input (id, π)	*store* (id, ν)

[5]We can also use the explicit salt scheme such that $\upsilon = h_1(\tau, \pi)$ where $\tau \in_R \{0,1\}^{t(k)}$. See later part.

$x_1, x_2 \in_R Z_q$		
$\upsilon = h_1(id, Bob, \pi)$		
$\mathcal{G}_0 = x_1 + \upsilon \bmod q$		
$\mathcal{G}_1 = g^{x_2}$	$\xrightarrow{id,\ x_1+\upsilon,\ g^{x_2}}$	
		$fetch\ (id, \nu)$
		$y \in_R Z_q$
	$\xleftarrow{(g^{x_2}\nu)^y}$	$\mathcal{G}_2 = (\mathcal{G}_1\nu)^y$
$e = h_2(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$		$e = h_2(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$
$\chi = (x_2 - \upsilon)^{-1}(x_1 + ex_2) \bmod q$		
$\alpha = (\mathcal{G}_2)^\chi$		$\beta = (g)^{\mathcal{G}_0 y}(\nu)^y(\mathcal{G}_1)^{ey}$
$\mathcal{K}_1 = h_3(\alpha)$		$\mathcal{K}_2 = h_3(\beta)$
$\mathcal{H}_{11} = h_4(\mathcal{G}_0, \mathcal{G}_1, \mathcal{K}_1)$	$\xrightarrow{h(g^{x_2}, \mathcal{K})}$	$\mathcal{H}_{12} = h_4(\mathcal{G}_0, \mathcal{G}_1, \mathcal{K}_2)$
		$verify\ \mathcal{H}_{11} \doteq \mathcal{H}_{12}$
$\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1)$	$\xleftarrow{h((g^{x_2}\nu)^y, \mathcal{K})}$	$\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$
$verify\ \mathcal{H}_{21} \doteq \mathcal{H}_{22}$		

Figure 5. AMP^{++} Protocol

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

1. *Alice* computes $\upsilon = h_1(id, Bob, \pi)$, $\mathcal{G}_0 = x_1 + \upsilon \bmod q$, 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 ν, 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 $\mathcal{G}_2 = (\mathcal{G}_1\nu)^y = g^{(x_2-\upsilon)y}$. He sends $\mathcal{G}_2$ to *Alice*.

3. After receiving message 2, *Alice* computes $e = h_2(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\chi = (x_2 - \upsilon)^{-1}(x_1 + ex_2) \bmod q$, and $\alpha = (\mathcal{G}_2)^\chi$. Note that $\alpha = (g^{(x_2-\upsilon)y})^{(x_2-\upsilon)^{-1}(x_1+ex_2)} = g^{y(x_1+ex_2)}$. 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 $e = h_2(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$, $\beta = (g)^{\mathcal{G}_0 y}(\nu)^y(\mathcal{G}_1)^{ey}$, $\mathcal{K}_2 = h_3(\beta)$ and $\mathcal{H}_{12} = h_4(\mathcal{G}_0, \mathcal{G}_1, \mathcal{K}_2)$. Note $\beta = g^{(x_1+\upsilon)y}g^{-\upsilon y}g^{x_2 ey} = g^{(x_1+ex_2)y}$. 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 υ (actually, π), 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 $\mathcal{H}_{12} = \mathcal{H}_{12}$, *Alice* also agrees on $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$ with authenticating *Bob* who knows ν.

It is clear to see $\mathcal{G}_2$ and an agreed key are unrelated 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 α^a does on the average through the simultaneous multiple exponentiation method[35, 28]. 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^{++}. For efficiency ($3E$), we considered the implicit salt scheme. However, we can accommodate the explicit 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* should compute β after receiving message 3. The protocol loses the parallel computation of α and β so that the parallel exponentiation cost is to be $4E$ rather than $3E$. This is the worst case in our AMP family.

5 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, SNAPI-X, AuthA and PAK-X[8, 19, 39, 27, 6, 10]. AMP implies the original protocol and other extensions if we do not explicitly notify.

5.1 Security of AMP

1. AMP provides perfect forward secrecy via the Diffie-Hellman problem and the discrete logarithm problem. That is, even if π (or v) is compromised, *Eve* cannot find old session keys because she is not able to solve the hard problems. We examined this feature in item 2 of Lemma 2.

2. Denning-Sacco attack is the case that *Eve*, who compromised an old session key, attempts to find π or to make the oracle accept her[11]. For the purpose, *Eve* has to solve the discrete logarithm problem even if $g^{(x+e)y}(= \alpha = \beta)$ is compromised. It is also infeasible to check the difference between e and v in $g^{(x+e)y}$ and $g^{(x+v)y}$ without solving the discrete logarithm of g^x. Therefore, AMP is secure against this attack. We examined this feature in item 1 of Lemma 2.

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 corresponds 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. Condition 2 and 5 of Theorem 2 supported this feature.

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. Actually, *Eve* is able to perform the on-line attack to either party but its failure is countable. Impersonation of the party or man-in-the-middle attack is also infeasible without knowing v or ν. Item 1 of Lemma 1 and Condition 2 and 4 of Theorem 2 directly handled the detectable on-line attacks, while the other items of Lemma 1 and Condition 5 of Theorem 2 handled the off-line frustration.

6. Off-line guessing attack is also infeasible because *Eve* cannot disintegrate $\mathcal{G}_2$. Condition 3 and 5 of Theorem 2 handled this feature. 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 $y^{'} = (x + v)y(x + v^{'})^{-1} \bmod q$ for guessed passwords without knowing both x and y. These features are examined in Lemma 1 and Condition 4 of Theorem 2.

7. Security against password-file compromise is the basic property of AMP except AMP^n that has a naked assumption. We exaimed this feature in item 2 of Lemma 1.

8. Information leakage is not an issue in AMP by virtue of random oracles for v and e.

5.2 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^{++}, the size of $\mathcal{G}_0$ can be bounded by $l(k) + \epsilon$ with negligible ϵ 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 $E(Alice : Bob)$. Note that AMP has intrinsically only $3E$, except that AMP^{++} has $4E$

with explicit salt. AMP has $E((g)^x : -)$, $E(- : (\mathcal{G}_1^y \nu^y))$ and $E((\mathcal{G}_2)^\chi : (\mathcal{G}_1^y g^{ey}))$ while all variants except AMP^{++} with explicit salt have similar operations. Here '$-$' means no-modular-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 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 family except for AMP^{++}. *Alice* can reduce her run time exponentiations to only one and parallel exponentiations to only two, by pre-computation of g^x. AMP^{++} needs *Alice* to generate two random numbers.

5. $\mathcal{G}_2$ can benefit from the simultaneous exponentiation method as $(\mathcal{G}_1)^y \nu^y$. β of AMP^{++} must benefit from the simultaneous exponentiation method for efficiency. As we mentioned already, $\alpha^a \beta^b$ needs 16% and $\alpha^a \beta^b \gamma^c$ needs 25% more multi-precision multiplications than α^a does on the average[35, 28].

6. In step 3, *Alice* should compute $(y + v)^{-1}$ but only in the q-order subgroup. Modular inversion, $O((\log q)^2)$, is much less expensive than modular exponentiation, $O((\log p)^3)$. Moreover, the size of q can be bounded by only $l(k) + \epsilon$ with negligible ϵ by virtue of a secure prime. Note that $O(\log l(k)) << O(\log p)$. Therefore, it is quite negligible when we consider modular exponentiation.

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 be a patent restriction on the elliptic curve cryptographic algorithms.

A rigorous efficiency comparison to the other protocols will be done in section 5.3

CONSTRAINTS. AMP gives very light constraints as follows.

1. AMP prefers g to be a generator of the large ($> l(k)$) prime-order subgroup Z_q for defeating and avoiding a small subgroup confinement effectively by confining exponentials into the large prime-order subgroup[30]. A secure prime modulus is higly 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[24].

2. A compromise of ν allows a guessing attack or a server impersonation but it is an inevitable feature of all verifier-based protocols[39]. As one of those protocols, AMP needs an additional guessing attack complexity for a client impersonation.

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

Table 1: Comparisons of Verifier-based Protocols

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.

5.3 Comparisons to Others

AMP is compared to the existing verifier-based protocols such as A-EKE, B-SPEKE, SRP, GXY, SNAPI-X, AuthA and PAK-X[8, 19, 39, 22, 27, 6, 10]. They are all wonderful protocols for password authentication research. We disregard the security issue because all of them are believed secure.

Table 1 compares them with regard to several factors such as the number of protocol steps, large message blocks, and exponentiations. The number of random numbers 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. Note that AuthA and PAK-X are the versions that use implicit salt. They must have five steps rather than three for accommodating explicit salt.

As we can see in Table 1, AMP is 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 paralle exponentiation for explicit salt). Comparisons of AMP^n to the related protocols such as EKE, SPEKE, SNAPI, PAK[7, 18, 27, 10] under their naked assumption, are skipped in this document. We give some basic introductions to those related protocols and discuss the superiority of AMP.

A-EKE. Bellovin and Merrit introduced the first verifier-based protocol, A-EKE in 1993 by augmenting their famous protocol DH-EKE[8]. A-EKE used symmetric encryption and digital signature[29] for authenticaed key exchange so that it had to have several constraints. However, A-EKE gave many researchers the right way to go for user authentication. Actually our study is also based on DH-EKE and A-EKE. or p-NEW signature[29].

B-SPEKE. Jablon proposed B-SPEKE, the verifier-based augmentation of SPEKE, in 1997[18, 19]. Jablon's paper includes very informative introductions to password authenticated key exchange as an addition to the famous work of EKE[18, 7]. A-SPEKE was an A-EKE based extension of SPEKE while B-extension was better for SPEKE than A-extension[19]. Among the set of B-SPEKE, the combined B-SPEKE was the most optimized so that it was compared in Table 1.

SRP. Wu proposed SRP that was notable in its practical approach, in 1998[39]. The benchmark showed its superiority to the earlier schemes[39]. SRP was not favorable to the generalization, especially in the elliptic curve group.

GXY. Kwon and Song proposed GXY that was derived from SRP for agreeing on the Diffie-Hellman exponential g^{xy}, in 1999[22]. However, the protocol was actually less efficient than SRP. GXY was also not favorable to the generalization.

SNAPI-X. MacKenzie and Swaminathan introduced the first provable verifier-based protocol, SNAPI-X, in 1999[27]. It was the augmented version of their basic protocol SNAPI in the same paper. It was notable in its full proof of security by combining RSA and Diffie-Hellman scheme at the cost of protocol efficiency.

AUTHA. Bellare and Rogaway introduced a provable approach, AuthA, by assuming an ideal cipher in 2000[6]. Their pure password version, MA-EKE2[5], was also notable in its provable approach. They give us the theoretical way for proof as always. The verifier-based version, AuthA, reduced her protocol steps with implicit salt. Coincidentally, its exponentiation sequences were almost the same to GXY[22]. We carefully point out that AuthA and MA-EKE2 could be vulnerable to password guessing due to $E_{f_1(\pi)}(g^x)$, $E_{f_2(\pi)}(g^y)$ and $h(h(A, B, g^x, g^y, g^{xy}), 1)$ if an agreed secret g^{xy} is compromised.

PAK-X. Boyko, MacKenzie, and Patel introduced the provable verifier-based protocol, PAK-X, in 2000[10]. PAK-X was notable in its clearly provable approach. PAK-X was the last protocol in their three kinds of protocols but the only one that used a verifier. It also reduces the number of protocol steps without exchanging salt but may have the lowest performance.

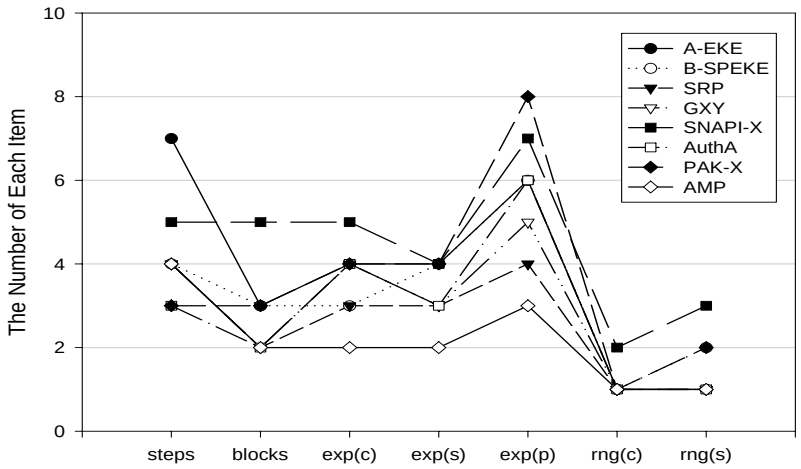

Figure 6. Graphical Representation of Table 1

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

1. AMP is a secure verifier-based protocol[6] on the basis of the amplified password proof 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 accommodating 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 under the amplified password as $(g^y)^\phi$ while *Alice* obtains it by $(g^{y\phi})^{\phi^{-1}}$ and raises it to x with random number e, i.e. $g^{y(x+e)}$.

6. AMP has a simple structure so that it is easy to understand and implement the protocol.

7. AMP is favorable to upgrading the existing system; AMP accommodates any kinds of salt schemes without notable degrading of the protocol performance so that the existing password file of various systems can be upgraded easily.

[6]Only AMP^n is a pure password-based protocol so that it is also comparable to the well-known related protocols such as EKE, SPEKE, SNAPI, PAK[7, 18, 27, 10] under the naked assumption such that the protocol is vulnerable to the password file compromise.

6 Applications

As one of verifier-based protocols, AMP can be used for user authentication and further confidential communications over the Internet. We give several examples for it.

6.1 A-Telnet and A-FTP

It is typical to apply the protocol to the existing remote authentication framework such as Telnet and FTP. For example, SRP already provides its enhanced Telnet and FTP versions. We can consider A-Telnet (AMPlified Telnet) and A-FTP (AMPlified FTP) for the same purpose with more flexibility and efficiency.

6.2 AA-Gate

Today every Internet user owns a number of accounts for various kinds of Internet sites. The users are mostly authenticated by giving their ids and passwords while it is still a far story for them to have their own private-key and certificate pairs for SSL (Secure Socket Layer). A small number of sites prevent eavesdropping of authentication information over SSL. It is hard for users to remember a large number of ids and passwords for each site. Therefore, all users prefer to choose their ids and passwords consistently and coherently. We should note that this feature makes simple but critical security holes on the Internet. If an account list of one large site is disclosed accidentally or intentionally, or the authentication information is eavesdropped for a specific account, then the other sites are vulnerable to tiny-dictionary attacks. This is critical even if the other site communicates with the browser over SSL on authentication phase.

We can construct AA-Gate (AMPlified Authentication Gate) for user convenience and enhanced security. The basic structure of AA-Gate is contrary to that of RADIUS (Remote Authentication Dial-In User Service). Every user opens his or her account at AA-Gate and registers his or her preferred sites on AA-Gate. Users can enter their old account information or utilize a password generator provided by AA-Gate. When users log on to AA-Gate, they can be securely guided to the registered sites by AA-Gate. Every communication between users and AA-Gate is secured by AMP while AA-Gate and the registered sites can communicate over SSL. After the successful authentication, a user can choose a registered site that she wants to visit. Then, the AA-Gate server downloads a proxy certificate to the client that acts furtherly for activating SSL session between the user and the server. The AA-Gate client (applet, or plug-in application) must communicate with the AA-Gate server again because it cannot possess the proxy private-key. Finally, users and the registered site can open their SSL session securely supported by AA-Gate's proxy certificate mechanism. A user does not need to own his or her certificate rather utilizes the proxy certificate of AA-Gate for opening the mutually authenticated SSL session. We can consider AA-Gate-User-Agent for conveniently registering a new site.

6.3 Networked Smart Card

A smart card is the best solution for enhancing the security on the Internet. Especially a crypto smart card is able to perform security operations as well as store the sensitive information such as a private-key. However, it is not easy to distribute such a card and a card-reader device to the existing infrastructure. Therefore, we can consider the Networked Smart Card over the Internet on the basis of AMP. This software and network based solution may expect the faster spread than hardware oriented solution.

We can consider two kinds of approaches for Networked Smart Card. One is a centralized approach and the other is a localized approach. Users must install their VSC (Virtual Smart Card) software on their machine. However, VSC can be provided as a Java applet in the centralized approach for users who are unwilling to install such a new software. Of course, every communication between the VSC and the SCS (Smart Card Server) must be secured by AMP. The sensitive information of users is stored in the SCS for the centralized approach, while the information is stored in the VSC for the localized approach.

6.4 A-RADIUS

The RADIUS protocol is a method of managing the exchange of authentication-related information in the network. The RADIUS remote access environment has three components such as Users, RAS (Remote Access Servers), and RS (RADIUS Server). The RADIUS protocol utilizes PAP (Password Authentication Protocol) or CHAP (Challenge Handshake Authentication Protocol) but all of them are vulnerable to password guessing attacks. Communication between RAS and RS are encrypted under a secret key but users and RAS communicates in the plain-text. We can consider A-RADIUS (AMPlified RADIUS) for enhanced security. For providing one-way authentication, only three messages of AMP are used for this architecture. All RAS has to do is relay three AMP messages between Users and RS, and wait for the authentication result from RS. It is still recommended to encrypt all messages between RAS and RS for authenticating RAS and securing internal conversation.

7 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 such as A-EKE, B-SPEKE and SRP. AMP has been designed on the basis of our simple idea, the amplified password proof. 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 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. For example, a private-key is not memorable for human users even in the public key infrastructure. Therefore, we might keep utilizing it over the Internet and in mobile environments even with the hardware-supported authentication schemes. A-Telnet, A-FTP, AA-Gate, Networked Smart Card, and A-RADIUS were only such examples.

Acknowledgment Author of this document thanks Mr. David Jablon for his helpful comment on this work.

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," Contribution to the IEEE P1363 study group for Future PKC Standards, available from *http://grouper.ieee.org/groups/1363/StudyGroup/Passwd.html#amp*

[24] 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

[25] 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

[26] S.Lucks, "Open key exchange: how to defeat dictionary attacks without encrypting public keys," The Security Protocol Workshop '97, April 7-9, 1997

[27] P.MacKenzie amd R.Swaminathan, "Secure network authentication with password identification," Presented to IEEE P1363a, August 1999

[28] A.Menezes, P.van Oorschot, S.Vanstone, *Handbook of applied cryptography*, CRC Press,Inc., 1997

[29] K.Nyberg and R.A.Rueppel, "Message recovery for signature scheme based on the discrete logarithm problem," Eurocrypt 94, pp. 182-193, 1994

[30] P.van Oorschot and M.Wiener, "On Diffie-Hellman key agreement with short exponents," EUROCRYPT 96, pp. 332-343, 1996

[31] S.Patel, "Number theoretic attacks on secure password schemes," IEEE Symposium on Security and Privacy, 1997

[32] 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

[33] J.Pollard, "Monte carlo methods for index computation mod p," Mathematics of Computation, vol.32, pp.918-924, 1978

[34] M.Roe, B.Christianson, D.Wheeler, "Secure sessions from weak secrets," Technical report from University of Cambridge and University of Hertfordshire, 1998

[35] C.P.Schnorr, "Efficient identification and signatures for smart cards," Crypto 89, LNCS, pp.239-251, 1989

[36] 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

[37] 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

[38] V.Voydoc and S.Kent, "Security mechanisms in high-level network protocols," Computing Surveys, vol.15, no.2, June 1983, pp.135-171

[39] T.Wu, "Secure remote password protocol," Internet Society Symposium on Network and Distributed System Security, 1998

Appendix : Genealogy of Password Protocol

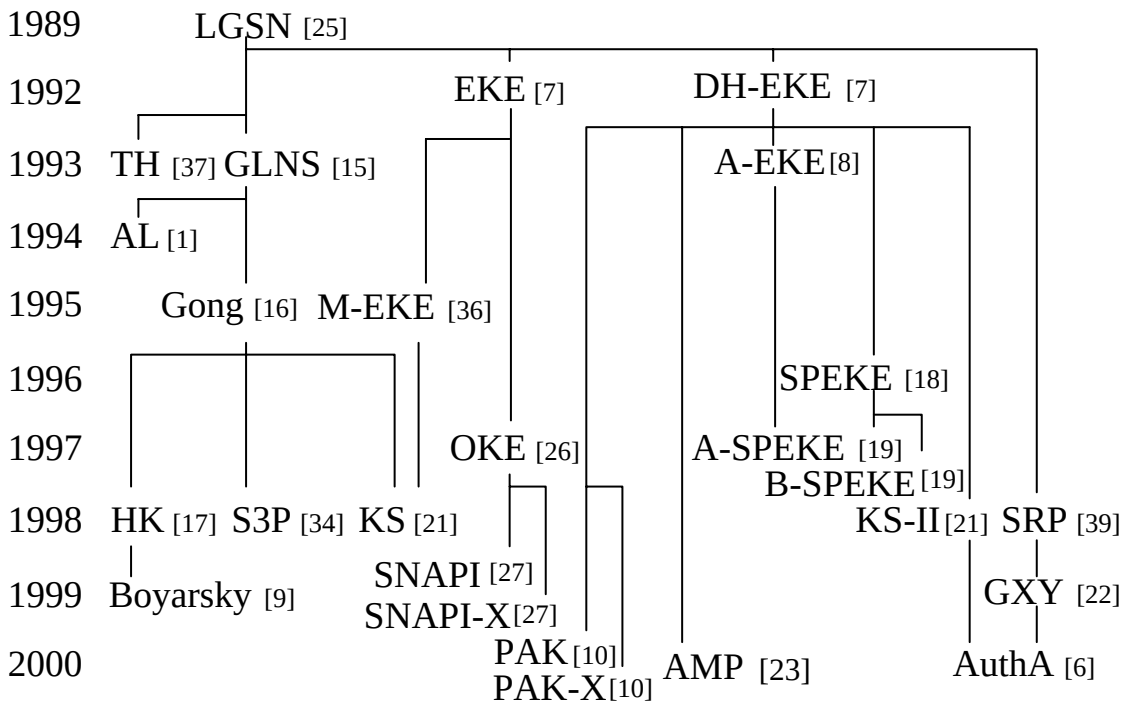

Figure 7. Password Authentication Protocols [7]

[7] 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.