Authentication and Key Agreement via Memorable Password

Taekyoung Kwon
University of California, Berkeley
tkwon@cs.berkeley.edu or ktk@emerald.yonsei.ac.kr

Updated on August 20, 2000
Preliminary Version 2.51 since May 1, 2000

Abstract

This paper presents a new password authentication and key agreement protocol, AMP, based on the amplified password idea. The intrinsic problems with password authentication are the password itself has low entropy and the password file is very hard to protect. We present the amplified password proof and the amplified password file for solving these problems. A party commits the high entropy information and amplifies her password with that information in the amplifed password proof. She never shows any information except that she knows it. Our amplified password proof idea is very similar to the zero-knowledge proof in that sense. We adds one more idea; the amplified password file for password file protection. A server stores the amplified verifiers in the amplified password file that is secure against a server file compromise and a dictionary attack. 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. In spite of those plentiful properties, AMP is actually the most efficient protocol among the related protocols due to the simultaneous multiple exponentiation method. Several variants such as AMP^i, AMP^n, AMP^{n+}, AMP$^+$, AMP^{++}, and AMP^c are also proposed. Among them, AMP^n is actually the basic protocol of this paper that describes the amplified password proof idea while AMP is the most complete protocol that adds the amplified password file. AMP^i simply removes the amplified password file from AMP. In the end, we give a comparison to the related protocols in terms of efficiency.

This manuscript is a preliminary version of our paper available from the IACR eprint archive, http://eprint.iacr.org/2000/026. 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]. It is available from the website, http://grouper.ieee.org/groups/1363/StudyGroup/Passwd.html#amp.

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

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 their 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; knowledge-based authentication, token-based authentication, and biometric authentication. Among them, the knowledge-based scheme is for human memory ($\approx$ mind). Actually, it is the most widely-used method due to such advantages as 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. However, a complex problem with this password-only authentication is that a human-memorable password has low entropy so that it could be vulnerable to malicious guessing attacks. The problem becomes much more critical in an open distributed environment. Moreover, password file protection is another problem that makes password authentication more unreliable. If a password file is compromised, at least it is vulnerable to dictionary attacks. A cryptographic protocol based on public-key cryptography is the most promising solution to this problem.

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. 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 while there was an earlier work describing the use of salt[38]. GLNS[15] was enhanced from LGSN. Due to the inefficiency and constraints of older schemes, various modifications and improvements have followed. They include TH[37], AL[1], M-EKE[36], Gong[16], KS[21], SPEKE[18, 19], S3P[34], SRP[39], HK[17], and GXY[22]. 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[26] introduced a provable approach and was followed by several great work such as SNAPI[27], EKE2[5], AuthA[6], and PAK[10]. They show the provable approach in this area is getting matured.

Among the password protocols, A-EKE, B-SPEKE, SRP, GXY, SNAPI-X, AuthA, and PAK-X are classified as password-verifier based protocols[8, 19, 39, 22, 27, 6, 10]. They allow the asymmetric model in which a client possesses a password, while a server stores its verifier rather than the password. Following A-EKE[8], B-SPEKE was augmented from SPEKE[18,

[1] Readers are referred to Figure 8 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.

19]. SRP showed efficient work on a verifier and GXY was derived from SRP[39, 22]. SNAPI-X was augmented from SNAPI while PAK-X was enhanced from PAK[27, 10]. AuthA was derived from several previous protocols but strongly studies a provable approach[6]. However, even the verifier-based protocols still allow dictionary attacks and server impersonation attacks if a server file is compromised. Currently, standardization work on this field is being considered in IEEE P1363 group.

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

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

For achieving the goal, we propose two simple ideas called the amplified password proof that makes a user amplify her password and prove to know the amplified password, and the amplified password file that makes a server store the amplified verifier for resisting a server file compromise and a dictionary attack. From the 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 major variants of AMP. They are called AMP^i, AMP^n, AMP^{n+}, AMP$^+$, AMP^{++}, and AMP^c. Among them, AMP^n is actually the basic protocol of this paper that describes the amplified password proof idea though it is designed for a symmetric setup. Several minor variants also can be considered but they are explained implicitly. We compare the efficiency of all verifier-based protocols. Actually, AMP is the most efficient protocols among the existing verifier-based protocols. Note that AMP will be designed to avoid explicit encryption/decryption at least for authentication and verification though it will provide a key agreement function.

2 AMP Protocol Design

2.1 Preliminaries

AMP is typically the two party case so that 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. λ 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. All hash functions are assumed to behave like random oracles for security proof[3]. Note 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 $h_i(Q_j)$, 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 these problems. We consider the multiplicative group Z_p^* and actually use its prime-order subgroup Z_q. 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 can be much smaller than p[30, 32, 33]. It is easy to make g by $\alpha^{(p-1)/q}$ where α generates Z_p^*. 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 party 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]. We strongly recommend to use a secure prime modulus because it allows much smaller q, e.g., closely down to $l(k)$.

2.2 Our Idea

Our idea is to "amplify" the low entropy of password (1) on the well-structured cryptographic protocol, i.e., on the secure interaction between communicating parties, and (2) in the well-structured password file, both for securing password authentication and password file.

DEFINITIONS. Firstly we give some useful definitions followed by our 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.*

We can consider two kinds of setup such as a symmetric setup and an asymmetric setup. The asymmetric setup could benefit 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 proof can be classified into the fully insecure password proof such as PAP(password only), the partially insecure password proof such as CHAP(challenge and handshake), 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.*

THE AMPLIFICATION. Our amplification idea is very simple, for example, *Alice* insists on her knowledge of password π by giving $x + \pi \bmod q$ rather than π only, while x is the randomly-chosen high entropy information. For the purpose, fresh x must be securely 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 ϖ defines a value that only who knows π and x can make from $\mathcal{A}(\pi; x)$ where x is chosen randomly at Z_q and π is a human-memorable password for an arbitrary amplification function $\mathcal{A}()$. (Note: ϖ 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 ϖ. 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 ϖ.*

$$
\begin{array}{rcl}
\text{Alice} & & \text{Bob} \\
\textit{initial commitment} & \stackrel{\mathcal{G}_1(x)}{\longrightarrow} & \\
 & \stackrel{\mathcal{G}_2(y)}{\longleftarrow} & \textit{challenge} \\
\textit{response} & \stackrel{\mathcal{H}(\varpi)}{\longrightarrow} &
\end{array}
$$

For secure commitment, $\mathcal{G}_1()$ should not reveal x even to *Bob*, for example, g^x. While $\mathcal{G}_2()$ transmits a fresh challenge, $\mathcal{H}()$ should imply the fact only that *Alice* knows ϖ without revealing any information about x and π. If we set $\mathcal{A}(\pi; x) = (x+\pi)^{-1} \bmod q$, then only who knows x and π can compute ϖ. So we set $\mathcal{G}_2() = g^{(x+\pi)y}$ for making verification information, i.e., $(g^{(x+\pi)y})^{\varpi} = g^y$. *Bob* who knows $\mathcal{G}_1()$ as well as g^π, can make $\mathcal{G}_2()$ by $(g^x g^\pi)^y$. As a result, both parties can get g^y, the verification information, so we set $\mathcal{H}() = g^y$. Of course, they can also make g^{xy} due to the Diffie-Hellman scheme. Therefore, we can derive the following theorem that is easy to prove by assuming x is randomly chosen at Z_q. ($hint:$ ϖ is not derivable from g^x, $g^{(x+\pi)y}$, g^y and even g^π. π as well as x are necessary for computing $\mathcal{A}(\pi; x)$.)

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

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 actually 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. For key exchange, *Alice* can derive a session key from g^{xy} and show the fact of agreeing on it. *Bob* is also able to do the same thing. A strong one-way hash function must be the best tool for this. For *Alice* to agree on g^{xy}, we set $\varpi = (x+\pi)^{-1}x \bmod q$. For mutual key confirmation as well as mutual authentication, however, the protocol must be configured by four steps to add *Bob*'s response. The following describes the basic version. Note that the cases, $x \in \{0,1\}^1$, $y \in \{0,1\}^1$, $\mathcal{G}_1 \in \{0,1\}^1$, $\mathcal{G}_2 \in \{0,1\}^1$, and their small subgroup confinements must be avoided for a security reason.

$$
\begin{array}{rcl}
\text{Alice}(id, \pi) & & \text{Bob}(id, g^\pi \ or \ \pi) \\
x \in_R Z_q & & \\
\mathcal{G}_1 = g^x & \stackrel{id, g^x}{\longrightarrow} & \\
 & & fetch\ (id, \pi) \\
 & & y \in_R Z_q \\
\varpi = (x+\pi)^{-1}x \bmod q & \stackrel{g^{(x+\pi)y}}{\longleftarrow} & \mathcal{G}_2 = (\mathcal{G}_1 g^\pi)^y \ or \ (\mathcal{G}_1)^y g^{\pi y} \\
\alpha = (\mathcal{G}_2)^{\varpi} & & \beta = (\mathcal{G}_1)^y
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{K}_1 = h_1(\alpha) & & \mathcal{K}_2 = h_1(\beta) \\
\mathcal{H}_{11} = h_2(\mathcal{G}_1, \mathcal{K}_1) & \stackrel{h(\mathcal{G}_1,\mathcal{K}_1)}{\longrightarrow} & \mathcal{H}_{12} = h_2(\mathcal{G}_1, \mathcal{K}_2) \\
 & & verify\ \mathcal{H}_{11} \doteq \mathcal{H}_{12} \\
\mathcal{H}_{21} = h_3(\mathcal{G}_2, \mathcal{K}_1) & \stackrel{h(\mathcal{G}_2,\mathcal{K}_2)}{\longleftarrow} & \mathcal{H}_{22} = h_3(\mathcal{G}_2, \mathcal{K}_2) \\
verify\ \mathcal{H}_{21} \doteq \mathcal{H}_{22} & &
\end{array}
$$

Figure 1. AMP^n Protocol

Both parties compute exponentials as like the Diffie-Hellman scheme. The difference is that the random exponent of α and the base of $\mathcal{G}_2$ are tactfully transformed. We call this protocol AMP$^{\mathsf{n}}$(AMP-naked) because the protocol is actually vulnerable to client impersonations as well as server impersonations and dictionary attacks if a password file is compromised. *Bob* can store π rather than g^π and for this case $\mathcal{G}_2$ benefits from the simultaneous multiple exponentiation method in terms of efficiency[28, 35]. Above all, g^π must be kept secure because it is actually vulnerable to guessing attacks. So we propose an amplified password file idea for improving the security of the password file.

The Amplified Password File. As for the password file, an asymmetric setup is preferred because of the weakness of text-equivalence in a symmetric setup[8, 19, 39]. If a password file is compromised, it can be used directly for a client impersonation in the symmetric setup. However, the low entropy of password makes the password file vulnerable to dictionary attacks and server impersonation attacks even if each password is hashed or exponentiated in the asymmetric setup. For example, a verifier such as $\upsilon = h(\pi)$ is vulnerable to guessing attacks and sever impersonation attacks. For the password file protection, encryption can be considered but the key management and performance issue must be overcome. The amplified password file is a password file of which a record includes an amplified verifier.

Definition 7 *The Amplified Verifier ν defines a value that only who knows ς and τ can use for password verification where ς is chosen randomly at Z_q and τ is is chosen randomly at $\{0,1\}^k$. Set $\nu = g^{(\varsigma+\tau)^{-1}\upsilon}$ where $\upsilon = h(id, \pi)$. If $(\varsigma+\tau)^{-1} = 1$, ν is not the amplified verifier. (Note: ν is semi-permanent.)*

A record of the amplified password file is (id, τ, ν). The amplified password file is stored in an ordinary server system while ς must be stored in and supplied from a secure storage device such as a smart card. Ideally, ς should not leave such a secure device but a bottleneck may be a problem with a poor-performance device. Otherwise, we can devise a special hardware device for performing ς related operations with higher performance but practically ς may be loaded to the server's memory. However, even if ς resides in the server's run-time memory, the memory dump and its analysis are necessary for launching a server impersonation attack or a dictionary attack with the compromised password file. That is, the amplified password file itself is secure against such attacks if ς is secure; even if not, an adversary cannot directly

impersonate a server or launch dictionary attacks without memory dump. AMP will be the protocol that enables the amplified password ideas and the key agreement. The following subsection describes AMP.

2.3 Complete Protocol Description

We set $\varpi = (x + \upsilon)^{-1}(x + e)$ where $\upsilon = h_1(id, \pi)$ and $e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$.

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

1. *Alice* and *Bob* share g, p and q. *id* indicates precisely a user name (id) while *Alice* and *Bob* mean rather addresses.
2. *Alice* chooses $\pi \in_R \{0,1\}^{\omega(k)}$ and notify *Bob*, in an authentic and confidential manner (secure registration, off-line distribution with picture id proof).
3. *Bob* chooses $\tau \in_R \{0,1\}^k$ and stores $(id, \tau, \nu = g^{(\varsigma+\tau)^{-1}\upsilon})$ where $\upsilon = h_1(id, \pi)$. *Bob* should throw away π and υ.

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

Figure 2. AMP 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 τ and ν, and computes $\mathcal{G}_2 = (\mathcal{G}_1)^y \nu^{(\varsigma+\tau)y}$ by choosing $y \in_R Z_q$. This can be done by the simultaneous multiple exponentiation method. Note that $\mathcal{G}_2 = (\mathcal{G}_1)^y \nu^{(\varsigma+\tau)y} = (g^x g^\upsilon)^y$. He sends $\mathcal{G}_2$ to *Alice*.

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

Small Discussion. AMP passes four messages between *Alice* and *Bob* who agree on $g^{(x+e)y}$ while AMP$^{\mathsf{n}}$ agreed on g^{xy}. The password file compromise still needs memory dump and its analysis for getting ς. The existence of e is explicit against a compromise of both. For example, if *Eve* sends $\nu^{(\varsigma+\tau)x}$ to *Bob*, then *Bob* will respond with $(g^{\upsilon x} g^\upsilon)^y$ and compute $\beta = (g^{\upsilon x} g^e)^y = g^{(\upsilon x+e)y}$. If e does not exist, β will be equal to $g^{\upsilon xy}$ so that it will be computable by $\mathcal{G}_2^{(x+1)^{-1}x}$. Due to the existence of e, *Eve* cannot compute β from ν, x, e and $g^{\upsilon y} (= \mathcal{G}_2^{(x+1)^{-1}})$. That is, she cannot falsely convince *Bob* that she is *Alice* even with the password file and ς. Here, we should note that the statistical difference between $\mathcal{G}_2$ and $\alpha(= \beta)$ is only dependent upon the difference between υ and e. If $ABS(\upsilon - e) = 0$, they must be exactly same to each other though its probability is extremely low. Such a case can be detected easily by both parties, for example, $\alpha \doteq \mathcal{G}_2$ and $\beta \doteq \mathcal{G}_2$. Note that υ and e is unchangeable and untraceable in $\mathcal{G}_2$ and $\alpha(= \beta)$ actually without knowing either x or y (see Lemma 1 in Appendix). We can make $h_1()$ and $h_2()$ have far difference for υ and e. In this point, we could set q larger than $h_2()$ where $h_2()$ should be at least k larger than $h_1()$. For example, we define $h_2(x) = h(01|x|01)h(10|x|01)$ while $h_1(x) = h(00|x|00)$ by recommending SHA-1 or RIPEMD-160 for 160-bit hash. Otherwise, there are several variants removing such a point, for example, AMP^n, AMP$^+$, and AMP^{++}. Final two steps can be modified, for example, $\mathcal{H}_{11} = h_4(\alpha, \mathcal{G}_1, \mathcal{G}_2, id, A, B)$ and $\mathcal{H}_{22} = h_5(\beta, \mathcal{G}_2, \mathcal{G}_1, B, A, id)$. We can choose salt τ implicitly by computing $f(id, B)$ where $f()$ is an implicit salt function[6, 10], for example,

$\upsilon = h_1(id, f(id, B), \pi)$. For updating the existing system such as Unix, we can modify υ such that $\upsilon = h(\tau', \pi)$ and sends τ' in message 2 where $h(\tau', \pi)$ is an existing verifier.

3 AMP Protocol Variants

We present five explicit variants of AMP and AMP$^{\mathsf{n}}$. They are AMP^i, AMP^{n+}, AMP$^+$, AMP^{++}, and AMP^c. Among them, AMP^i excludes the use of the amplified password file while AMP^{n+} loses the zero-knowledge property for enabling verifier-based authentication to AMP$^{\mathsf{n}}$. AMP$^+$ and AMP^{++} are extended for perturbing the structural similarity between $\mathcal{G}_2$ and $\alpha(=\beta)$ in AMP. Every AMP provides one-way authentication in three steps. Finally, AMP^c is a crippled version of AMP family for one-way authentication in two steps.

3.1 AMP^i

AMP^i excludes the amplified password file from AMP. The difference of protocol setup is that *Bob* stores $(id, \tau, \nu = g^{\upsilon})$ where $\upsilon = h_1(\tau, \pi)$. Of course, salt can be obtained implicitly. The difference in protocol run is that *Alice* computes the inverse of the amplified password, after receiving message $\mathcal{G}_2$. AMP^i slightly reduces the running time about $\mathcal{G}_2$ but loses the strong security against a password file compromise. That is, if the password file is compromised, *Eve* is able to directly impersonate a server or launch dictionary attacks in AMP^i while it was much more difficult in AMP. We set $\varpi = (x + \upsilon)^{-1}(x + e) \bmod q$.

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

$$\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1) \qquad \xleftarrow{h(\mathcal{G}_2,\mathcal{K}_2)} \qquad \mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$$
$$verify\ \mathcal{H}_{21} \doteq \mathcal{H}_{22}$$

Figure 3. AMP^i Protocol

3.2 AMP^{n+}

AMP^n was totally vulnerable to the password file compromise. However, we can extend AMP^n for verifier-based authentication in the asymmetric setup model. Comapred to AMP, this extension is proposed for efficiency and "easy deployment" at the cost of security. AMP^{n+} actually loses the zero-knowledge property, that is, *Bob* is always able to read υ in a protocol run. The main difference of protocol setup is that $\nu = h_2(\tau, \upsilon)$ where $\upsilon = h_1(id, \pi)$. Of course, we can remove salt or use implicit salt; the following version explicitly uses salt. For AMP^{n+}, we define a function such that $\mathcal{E}(x, y) = x + y \bmod q$ and $\mathcal{D}(x, y) = x - y \bmod q$. We can replace their operations with a modular multiplication or a conventional encryption function though the conventional encryption is not recommended. We set $(x+\nu)^{-1}x \bmod q$.

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

$$\begin{array}{rcl}
\text{Alice}(id, \pi) & & \text{Bob}(id, \tau, \nu) \\
x \in_R Z_q & & \\
\mathcal{G}_1 = g^x & \xrightarrow{id, g^x} & \\
 & & fetch\ (id, \tau, \nu) \\
\upsilon = h_1(id, \pi) & & y \in_R Z_q \\
 & \xleftarrow{\tau, (g^{(x+\nu)y}} & \mathcal{G}_2 = (\mathcal{G}_1)^y g^{\nu y} \\
\nu = h_2(\tau, \upsilon) & & \\
\varpi = (x+\nu)^{-1}x \bmod q & & \\
\alpha = (\mathcal{G}_2)^{\varpi} & & \beta = (\mathcal{G}_1)^y \\
\mathcal{K}_1 = h_3(\alpha) & & \mathcal{K}_2 = h_3(\beta) \\
e = h_4(id, Alice, Bob, \mathcal{K}_1, \alpha) & & e = h_4(id, Alice, Bob, \mathcal{K}_2, \beta) \\
\mathcal{H}_{11} = \mathcal{E}(e, \upsilon) & \xrightarrow{\mathcal{E}(e,\upsilon)} & \\
 & & verify\ \nu \doteq h_2(\tau, \mathcal{D}(\mathcal{H}_{11}, e)) \\
\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1) & \xleftarrow{h(\mathcal{G}_2,\mathcal{K}_2)} & \mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2) \\
verify\ \mathcal{H}_{21} \doteq \mathcal{H}_{22} & &
\end{array}$$

Figure 4. AMP^{n+} Protocol

e^{-1} is applied to $\mathcal{H}_{11}$ for recovering υ.

3.3 AMP$^+$

AMP$^+$ perturbs the structural similarity between $\mathcal{G}_2 = g^{(x+\upsilon)y}$ and $\alpha = \beta = g^{(x+e)y}$. The difference in protocol setup is to set $\varpi = (x+\upsilon)^{-1}(x+e_2) \bmod q$. The main difference in protocol run is that both parties compute e_1 such that $e_1 = h_2(\mathcal{G}_1, id, Alice, Bob)$. We set $\varpi = (xe_1+\upsilon)^{-1}(x+e_2) \bmod q$. The randomness of e_1 is totally dependent upon that of $\mathcal{G}_1$ so that *Bob* cannot contribute to its randomness, while the randomness of e_2 is dependent upon that of $\mathcal{G}_2$ as well as $\mathcal{G}_1$.

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

Figure 5. AMP$^+$ Protocol

Note that $\mathcal{G}_2 = g^{(xe_1+\upsilon)y}$ while $\alpha = g^{(x+e_2)y}$. We should define $h_6(x) = h(10|x|01)$ for AMP$^+$.

3.4 AMP^{++}

AMP^{++} is another form of perturbing the structural similarity between $\mathcal{G}_2$ and $\alpha(=\beta)$. The difference in protocol setup is that *Bob* stores $(id, \nu = g^{-(\varsigma+\tau)^{-1}\upsilon})$ where $\upsilon = h_1(id, f(id, Bob), \pi)$. Protocol run is different that *Alice* chooses two ephemeral parameters such as x_1 and x_2. *Alice* computes $\mathcal{G}_0 = x_1 + \upsilon \bmod q$ and $\mathcal{G}_1 = g^{x_2}$, sends them to *Bob* who will respond with $\mathcal{G}_2$. We set $\varpi = (x_2 - \upsilon)^{-1}(x_1 + ex_2) \bmod q$ so that *Alice* computes $\chi = \varpi^{-1}(x_1 + ex_2) \bmod q$ for

getting α. *Bob* gets β by computing $(g)^{\mathcal{G}_0 y}(\nu)^y(\mathcal{G}_1)^{ey}$. Therefore, the agreed key is $g^{(x_1+ex_2)y}$ while $\mathcal{G}_2 = g^{(x_2-\upsilon)y}$.

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

Figure 6. AMP^{++} Protocol

3.5 AMP^c

AMP^c is a crippled version of AMP for allowing one-way authentication in a very restricted environment where *Bob* is allowed only one random challenge and *Alice* is allowed only one response to it. This protocol is inevitably vulnerable to a server impersonation attack. However, our idea is that if a server is strongly wired, AMP^c may be useful for strong authentication even without encryption in two steps. For example, practically, by combining with a digital signature scheme, AMP^c can be a complete protocol like sf AMP only in two steps. We define a signature function $\mathcal{S}()$ and a verification function $\mathcal{V}()$ for AMP^c. We assume *Alice* has a certificate of *Bob* but it can be sent to *Alice* in step 1. The protocol setup is the same to that of AMP. We set $\varpi = ye + \upsilon \bmod q$ where $\upsilon = h_1(id, \pi)$.

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(id, π)		Bob(id, ν)
		$x \in_R Z_q$
		$\mathcal{G}_1 = g^x$
	$\xleftarrow{\mathcal{G}_1, \mathcal{S}_1}$	$\mathcal{S}_1 = \mathcal{S}(\mathcal{G}_1)$
$verify\ \mathcal{D}(\mathcal{S}_1)$		
$y \in_R Z_q$		
$\mathcal{G}_2 = g^y$		
$e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$		
$\upsilon = h_1(id, \pi)$		
$\varpi = ye + \upsilon \bmod q$		
$\alpha = (\mathcal{G}_1)^{\varpi}$		
$\mathcal{H}_1 = h_3(\alpha)$	$\xrightarrow{id, \mathcal{G}_2, \mathcal{H}_1}$	
		$e = h_2(\mathcal{G}_1, \mathcal{G}_2, id, Alice, Bob)$
		$fetch\ (id, \nu)$
		$\beta = (\mathcal{G}_2)^{xe} \nu^{(\varsigma+\tau)x}$
		$\mathcal{H}_2 = h_3(\beta)$
		$verify\ \mathcal{H}_1 \doteq \mathcal{H}_2$

Figure 7. AMP^c Protocol with Digtal Signature

The above protocol is an implicit salt version of AMP^c but it is easy to get a symmetric setup version by removing e where a server stores π or υ. Of course, an explicit salt version must need one more step for exchanging the assigned salt. AMP^c with digital signature scheme must be useful for secure web login by assuming *Alice* a signed active program such as a signed Java Applet. Suppose that *Alice* is a signed Java Applet that can be downloaded from a web server *Bob*; the signed applet is being widely used for preventing a hostile applet. Then, our simple idea is embedding an uncertified signature-verification key (a public key) of *Bob* into *Alice* because *Alice* is an active program signed by a certificate authority. That is, we don't need to verify *Bob*'s verification key additionally. Otherwise, we can remove the digital signature of $\mathcal{G}_1$ by simply imbedding g^c in the singed Java Applet where c is a private key of a server.

4 Analysis and Comparison

4.1 Security of AMP

Appendix A (Lemma 1,2 and Theorem 2) may be helpful for security discussion.

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

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 υ in $g^{(x+e)y}$ and $g^{(x+\upsilon)y}$ without solving the discrete logarithm of g^x. Therefore, AMP is secure against this attack (Lemma 2-1).

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 (Theorem 2-2,5). 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.

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 (Lemma 1, Theorem 2). 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 υ or ν.

6. Off-line guessing attack is also infeasible because *Eve* cannot disintegrate $\mathcal{G}_2$ (Lemma 1, Theorem 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 $y' = (x+\upsilon)y(x+\upsilon')^{-1} \bmod q$ for guessed passwords without knowing both x and y.

7. Security against password-file compromise is the basic property of AMP family except AMP^n that has a naked assumption (Lemma 1-2). Additionally, AMP, AMP$^+$, AMP^{++}, and AMP^c provides the stronger security against the password file compromise by using ς without degrading the performance notably; they make such an attack much more difficult. That is, the password file compromise does not allow dictionary attacks and server impersonations. Even if ς is compromised, a dictionary attack is still necessary to impersonate a client.

4.2 Efficiency and Constraints

EFFICIENCY. We examine the efficiency of AMP and compare it with other related protocols.

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 $E(g^x : -)$, $E(- : (\mathcal{G}_1)^y \nu^{(\varsigma+\tau)y})$ and $E(\mathcal{G}_2^{\varpi} : \mathcal{G}_1^y g^{ey})$ while all variants have similar operations. Here '$-$' means no-modular-exponentiation such as $O((\log n)^3)$. Note that AMP operations should benefit from the simultaneous multiple exponentiation method for efficiency[35, 28]. As for ${g_1}^{e_1}{g_2}^{e_2}$, we don't need to compute ${g_1}^{e_1}$ and ${g_2}^{e_2}$ separately. A simple description of the simultaneous method is as follows;

   (a) `t = length(e); // length of exponent :  ⌊log q⌋ + 1`

   (b) $\mathtt{g_x} = \mathtt{g_1 g_2}$ `mod p; // precomputation`

   (c) `set() {G[0]=1; G[1]=`$\mathtt{g_1}$`; G[2]=`$\mathtt{g_2}$`; G[3]=`$\mathtt{g_x}$`;}`

   (d) `A = 1;`

   (e) `set() {for(i=1;i<=t;i++) {B`$_\mathtt{i}$ `= ExponentArray(i);}}`

   (f) `for(i=1;i<=t;i++) {A = A*A mod p; A=A*G[B`$_\mathtt{i}$`] mod p;}`

   (g) `return(A);`

   Note that ${g_1}^{e_1}{g_2}^{e_2}$ needs 16% and ${g_1}^{e_1}{g_2}^{e_2}{g_3}^{e_3}$ needs 25% more multi-precision multiplications than ${g_1}^{e_1}$ does on the average[35, 28].

3. Each party of AMP performs only two exponentiations, respectively, regarding the efficiency of the simultaneous multiple exponentiation. 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 once and parallel exponentiations to only twice, by pre-computation of g^x. AMP^{++} needs *Alice* to generate two random numbers.

5. In step 3, *Alice* should compute $(x+v)^{-1}$ 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.

	Protocol	*Large*	*Exponentiations*			*Random Numbers*	
	Steps	*Blocks*	*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	5 (+2) / **3** (+0)	**2** (+0)	4 (+2)	3 (+1)	6 (+3)	**1** (+0)	**1** (+0)
PAK-X	5 (+2) / **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

6. 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 algorithms.

Efficiency can be compared to the other related protocols such as A-EKE, B-SPEKE, SRP, GXY, SNAPI-X, AuthA and PAK-X[8, 19, 39, 22, 27, 6, 10]. Table 1 compares them with regard to several factors such as the number of protocol steps, large message blocks, and exponentiations. Note that the number of exponentiations of AMP is approximated; actually 2.32 for a server and 3.32 for parallel on the average. Note that AuthA and PAK-X have five steps with explicit salt and three steps with implicit salt. We consider five as their steps respectively because other protocols are the cases that use explicit salt. The use of explicit salt does not affect any other parameters of AuthA and PAK-X except the number of steps. The number of random numbers is given as a subsidiary reference. The number of parallel exponentiations could compare approximately the amount of protocol execution time. The value in parenthesis implies the difference from the most efficient one that is denoted by bold characters. Note that AMP provides the stronger security against the password file compromise compared to all the others in Table 1.

CONSTRAINTS. 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]. A compromise of ν does not allow a guessing attack

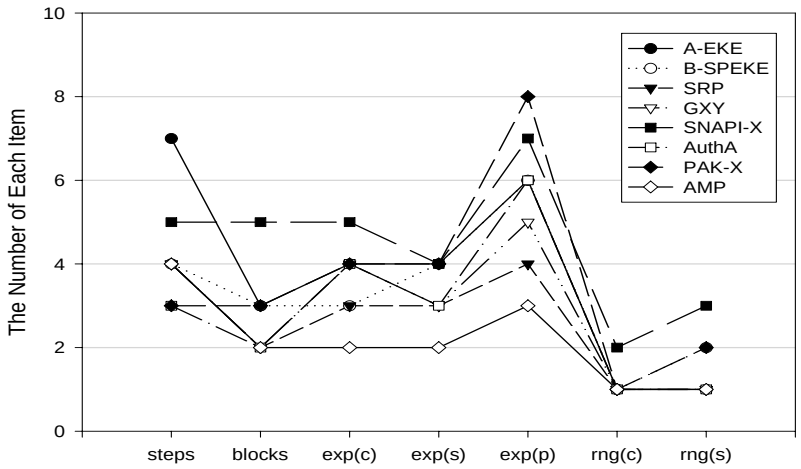

Figure 8. Graphical Representation of Table 1

and a server impersonation without additional comromise of ς that can be loaded to memory from a secure storage like a smart card. As a strong verifier-based protocol, AMP needs an additional guessing attack complexity for a client impersonation even if ν and ς are all compromised. 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.

4.3 Why AMP

Figure 8 rewrites Table 1 graphically so that we can see AMP($\diamond$) has the best performance.

1. AMP is a secure password(-verifier) based protocol on the basis of the amplified password proof and the amplified password file, and its security is provable in the random oracle model.

2. AMP is the most efficient protocol among the existing verifier-based protocols; AMP provides the best efficiency even with the amplified password file.

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 various functional considerations.

5. AMP truly allows the Diffie-Hellman based key agreement.

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 degrade of performance.

5 Conclusion

In this paper, we introduced a new protocol, AMP, for password authentication and key agreement, by following the various notable predecessors. AMP has been designed on the basis of the amplified password proof and the amplified password file ideas. The adavantages are well summarzied in section 4.3. We are considering several applications such as A-Updates (A-Telnet, A-FTP, A-RADIUS, A-CHAP, and etc.), A-Web Login, AA-Gate, and Networked Smart Card. 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 so that we need a hardware storage. We might keep using passwords over the Internet and in mobile environments even with the hardware-supported authentication schemes such as smart card or biometric method. Appendices include the proof story and the geneology of password protocols.

Acknowledgment The author thanks Doug Tygar, David Wagner, David Jablon, Radia Perlman and Li Gong for their helpful comments and kind suggestions 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," Eurocrypt 2000

[6] M.Bellare and P.Rogaway, "The AuthA protocol for password-based authenticated key exchange," 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," ACM Conference on Computer and Communication Security, 1999

[10] V. Boyko, P. MacKenzie and S. Patel, "Provably secure password authenticated key exchange using Diffie-Hellman," 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," ACM Symposium on Operating System Principles, 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

A AMP Proof Story

A.1 A Communication Model

We assume all communication among interacting parties is under the adversary's control[3].

THE PROTOCOL. The protocol can be formally specified by an efficiently computable function Π for two players; set $I \in \{A, B\}$ for *Alice* and *Bob*. *Eve* is not included in the players[3].

Definition 8 *Our protocol is a set of function* $\Pi_I(1^k, \sigma, \kappa, r) = (m, \delta, \mu)$ *for* I.

1^k is the security parameter, $k \in \mathcal{N}$. $\sigma \in \{0,1\}^*$ is the secret information of the sender. $\kappa \in \{0,1\}^*$ is the conversation so far. $r \in \{0,1\}^\infty$ is the random coin flips of the sender. $m \in \{0,1\}^* \cup \{*\}$ is the next message for a reply. $\delta \in \{Accept, Reject, *\}$ is the decision. $\mu \in \{0,1\}^* \cup \{*\}$ is the agreed session key. For example, Π_A^s indicate *Alice* computes on *Bob*'s message and gives out an output in session s. 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. Thus *Eve* can 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[2] 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

[2] Actually the LW-key, i.e., the password, is chosen by a human-user through a restricted input device.

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. *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. CompromisedUnfreshSession is negligible 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.

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: 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 $\sigma_E = \mathcal{W}(1^k, E, r_G)$. Choose a string $r_i^s \in_R \{0,1\}^\infty$ for $i \in I$ and $s \in \mathcal{N}$, and a string $r_E \in_R \{0,1\}^\infty$. Let $\kappa_i^s = \lambda$ for all $i \in I$ and $s \in \mathcal{N}$. 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 $\kappa_i^s = \kappa_i^s.n$. The adversary chooses an oracle Π_i^s and asks as she wants.

Security Definition. The following definition is derived from work of Bellare and Rogaway[3],

and the following work of Stefan Lucks[26].

Definition 9 *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 υ 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.

A.2 Security Examination

Note that we abbreviate ς for convenience in this section. 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 $\mathcal{G}_2 = g^{(x+\upsilon)y}$. *Eve* could find $\alpha^{'} = g^{y'(x+e)}$ by computing $\mathcal{G}_2{}^{(x+\upsilon')^{-1}(x+e)}$ and asking $\mathcal{Q}(B, s, guess)$ with $\upsilon^{'} \in_R \{0,1\}^{\omega(k)}$. 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 $Pr[\mathsf{Discrete\text{-}Log}^{E}(k)]$ 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 $y^{'} \equiv (x+\upsilon')y(x+\upsilon)^{-1} (\mathrm{mod}\ q)$. 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 $y^{'} \equiv y (\mathrm{mod}\ q)$. 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, $y^{'} \equiv (x+\upsilon')y(x+\upsilon)^{-1} (\mathrm{mod}\ q)$. Finding x from $\mathcal{G}_1$ is bounded by $Pr[\mathsf{Discrete\text{-}Log}^{E}(k)]$ 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, $(x+\upsilon^{'})y(x+\upsilon)^{-1}(x+e) \equiv (x+e)y + \upsilon^{'}y - \upsilon^{''}y (\mathrm{mod}\ q)$ is true, then she can find υ in $2^{-\omega(k)}$ probability, regardless of x and $\upsilon^{'}$. Otherwise, she has to find x first. We can rewrite it as, $(x+\upsilon^{'})\not{y}(x+\upsilon)^{-1}(\not{x}+\not{e}) \equiv (\not{x}+\not{e})\not{y}(1+\upsilon^{'}x^{-1} - \upsilon^{''}x^{-1}) (\mathrm{mod}\ q)$. That is, $(x+\upsilon^{'})(x+\upsilon)^{-1} \equiv (1+\upsilon^{'}x^{-1} - \upsilon^{''}x^{-1}) (\mathrm{mod}\ q)$. We can transpose $(x+\upsilon)^{-1}$ so that, $(x+\upsilon^{'}) \equiv (x+\upsilon) + (x+\upsilon)\upsilon^{'}x^{-1} - (x+\upsilon)\upsilon^{''}x^{-1} \equiv x + \upsilon + \upsilon' + \upsilon\upsilon^{'}x^{-1} - \upsilon^{''} - \upsilon\upsilon^{''}x^{-1}$ $\equiv (x+\upsilon^{'}) + (\upsilon - \upsilon^{''}) + (\upsilon^{'} - \upsilon^{''})\upsilon x^{-1} \pmod{q}$. Then, we can transpose $(x+\upsilon^{'})$ so that; $0 \equiv (\upsilon - \upsilon^{''}) + (\upsilon^{'} - \upsilon^{''})\upsilon x^{-1} (\mathrm{mod}\ q)$. Therefore, the equality such that, $\upsilon = \upsilon^{'} = \upsilon^{''}$ (due to $\upsilon = \upsilon^{''}$ and $\upsilon^{'} = \upsilon^{''}$), is the mandatory requiremet of this modular equation. However, the probability of success is negligible because $\upsilon \neq \upsilon^{'}$ with very high probability as we mentioned above in (a). Therefore, *Eve* must find x for getting υ. The probability of verifying $\alpha^{''} \doteq \alpha$ is $Pr[] < 2^{-k}$.

After all, the probability of success is negligible for forged queries in FreshSession. □

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

Proof Sketch: We deal with each condition of Definition 9.

1. A completeness of the protocol in MatchedSession is already shown in Figure 1.

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 $Pr[\mathsf{No\text{-}Matching}^E(k)] \leq 2^{-k}$.
(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 ϖ from $g^{\varpi y}$ and find y where $\varpi = x + \upsilon$. Finding ϖ includes x so that both findings must be bounded by $Pr[\text{Discrete-Log}^E(k)]$. Even if we assume ϖ 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*. □

B Genealogy of Password Protocol

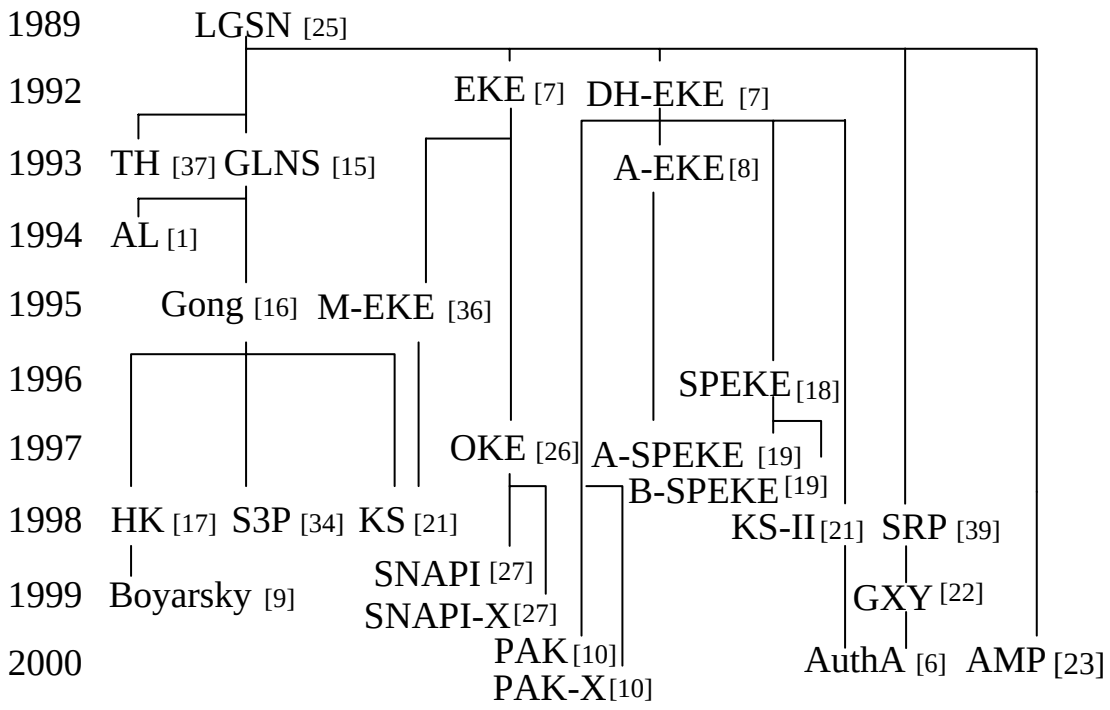

Figure 9. Password Authentication Protocols [3]

[3] 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.