Additional Notes to "Ultimate Solution to Authentication via Memorable Password" May 1, 2000 version

Taekyoung Kwon*

May 25, 2000

Abstract

This short letter adds informative discussions to our previous contribution, "Ultimate Solution to Authentication via Memorable Password"[1].

1 Introduction

Message 2 of AMP, $\mathcal{G}_2$, is structured that $\mathcal{G}_2 = (\mathcal{G}_1\nu)^y$ while the agreed secret key is $g^{xy}(= \alpha = \beta)$. Note that $\mathcal{G}_1 = g^x$ and $\nu = g^v$[1]. Therefore, a care must be taken in regarding how much information $\mathcal{G}_2(= g^{xy}g^{vy})$ could leak about g^{xy}, though the probability of $\nu^y = 1$ is very low. In this letter, we propose a method to avoid the argument about the information leakage by slightly modifying the protocol. Note that we abbreviate mod p.

Hints :

1. Let $\mathcal{G}_2$ be $(\mathcal{G}_1^e\nu)^y$ with random e.
2. Let parties agree on $g^{(x_1+ex_2)y}$ rather than g^{xy}.

2 Avoiding Information Leakage

We propose two extended protocols from original AMP for avoiding the information leakage argument. They are AMP$^+$ (hint 1) and AMP^{++} (hint 2).

Numerical Assumption. *Bob* chooses g which generates a prime-order subgroup Z_q where $p = qr + 1$. Note that a prime q must be sufficiently large ($> l(k)$)[1] to resist Pohlig-Hellman decomposition and various index-calculus methods but much smaller than p[5, 6, 7]. It is

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

easy to make g by $\alpha^{(p-1)/q}$ where α generates Z_p^*. 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 $p = 2q+1[3]$. However, we strongly recommend to use a secure prime modulus p. Such a modulus should makes our modified protocol secure and efficient.

2.1 AMP$^+$

Our first extention is AMP$^+$ based on hint 1. Protocol setup of AMP$^+$ is exactly the same to that of AMP so that its description is skipped here (see [1] for protocol setup). Note that Bob stores $(id, \tau, \nu = g^{\upsilon})$ where $\upsilon = h_1(\tau, \pi)$, $\tau \in_R \{0,1\}^{t(k)}$, and $\pi \in_R \{0,1\}^{\omega(k)}$.

PROTOCOL RUN. The following describes how to run AMP$^+$. Note that the cases, $x \in \{0,1\}^1$, $y \in \{0,1\}^1$, $g^x \in \{0,1\}^1$, $(g^{xe}\nu)^y \in \{0,1\}^1$, and their small subgroup confinements must be avoided for a security reason. *Alice* and *Bob* can easily detect and discard such insecure parameters in the protocol.

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

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 $e = h_2(\mathcal{G}_1, id, Alice, Bob)$ and $\mathcal{G}_2 = \mathcal{G}_1^{ey}\nu^y$ by choosing $y \in_R Z_q$. This can be done by the simultaneous exponentiation method. Note that $\mathcal{G}_2 = (g^{xe}\nu)^y = g^{(xe+v)y}$. He sends $(\tau, \mathcal{G}_2)$ to *Alice*.

3. While waiting for message 2, *Alice* computes $e = h_2(\mathcal{G}_1, id, Alice, Bob)$. After receiving message 2, *Alice* computes $v = h_1(\tau, \pi)$, $\chi = (xe + v)^{-1}x \bmod q$ and $\alpha = (\mathcal{G}_2)^\chi$. Note that $\alpha = (g^{(xe+v)y})^{(xe+v)^{-1}x} = g^{yx}$. 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 $\beta = (g^x)^y = g^{xy}$, $\mathcal{K}_2 = h_2(\beta)$ and $\mathcal{H}_{12} = h_4(\mathcal{G}_1, \mathcal{K}_2)$. After receiving message 3, *Bob* compares $\mathcal{H}_{12}$ with $\mathcal{H}_{11}$. If they are matched, then he computes $\mathcal{H}_{22} = h_5(\mathcal{G}_2, \mathcal{K}_2)$ and sends $\mathcal{H}_{22}$ to *Alice*. This means he authenticated *Alice* who knows v (actually, π), and agreed upon $\mathcal{K}(= \mathcal{K}_1 = \mathcal{K}_2)$.

5. While waiting for message 4 from *Bob*, *Alice* computes $\mathcal{H}_{21} = h_5(\mathcal{G}_2, \mathcal{K}_1)$. 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.

Whole structure of AMP$^+$ is exactly the same to that of AMP except that $\mathcal{G}_2$ includes a randomizer e for avoiding the information leakage of g^{xy}; only computing e has been added to AMP. Therefore, security and efficiency are approximately bounded by AMP. AMP$^+$ also passes four messages between *Alice* and *Bob*, and benefits from the simultaneous multiple exponentiation method; $\alpha^a\beta^b$ needs 16% more multi-precision multiplications than α^a does on the average[8, 4]. Therefore, each party's exponentiation number is still two while parallel exponenetiation is still $3E$. The randomness of e is totally dependent upon the randomness of g^x so that *Bob* cannot contribute to its randomness. The information leakage argument has been clearly avoided since $\mathcal{G}_2 = g^{(xe+v)y}$ while the agreed key is still g^{xy}. Now $\mathcal{G}_2$ is not related to the agreed key any more without solving $\log_g g^v$ and $\log_g g^y$. AMP$^+$ is a simple extention of AMP for avoiding the information leakage and is as practical as AMP.

2.2 AMP^{++}

Our second extention is AMP^{++} based on hints 2. AMP^{++} has a little different protocol setup procedure from AMP and AMP$^+$.

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 $\upsilon = h_1(id, Bob, \pi)$[1].

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, $x_1 \in \{0,1\}^1$, $x_2 \in \{0,1\}^1$, $y \in \{0,1\}^1$, $g^x \in \{0,1\}^1$, $(g^x\nu)^y \in \{0,1\}^1$, and their small subgroup confinements must be avoided for a security reason. *Alice* and *Bob* can easily detect and discard such insecure parameters in the protocol.

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

Figure 2. AMP^{++} Protocol

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

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 $y \in_R Z_q$.

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

This can be done by the simultaneous exponentiation method, i.e., ${\mathcal{G}_1}^y \nu^y$. Note that $\mathcal{G}_2 = (g^{x_2}\nu)^y = g^{(x_2-\upsilon)y}$. 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.

The information leakage argument has been clearly avoided since the agreed key is $g^{(x_1+ex_2)y}$ while $\mathcal{G}_2 = g^{(x_2-\upsilon)y}$. AMP^{++} also passes four messages between *Alice* and *Bob*, and benefits from the simultaneous multiple exponentiation method; $\alpha^a\beta^b$ needs 16% and $\alpha^a\beta^b\gamma^c$ needs 25% more multi-precision multiplications than α^a does on the average[8, 4]. Considering the benefit of the simultaneous method, it can be said that each party of our modified protocol needs the exponentiation, $O((\log n)^3)$, still for two times, respectively. In addition, the parallel exponentiation is still three times $(3E)$[1]. Several mod q operations and one mod p exponentiation in α must compensate for the mod p simultaneous exponentiation in β, and they are bounded by $O((\log n)^3) + \epsilon$ for negligible expense ϵ. The use of a secure prime allows $q(\geq l(k))$ to be 160 bits long. It is helpful for effciency in message size and Z_q operation.

Conventional Salt in AMP^{++}. For efficiency $(3E)$, we considered the salt scheme discussed in Bellare and Rogaway[2]. However, we can accommodate the conventional salt scheme at the cost of parallel exponentiation $(4E)$. Instead of *id*-based implicit salt, *Bob* chooses $\tau \in_R \{0,1\}^{t(k)}$ and stores $(id, \tau, \nu = g^{-\upsilon})$ where $\upsilon = h_1(\tau, \pi)$ on setup phase. *Bob* should send *Alice* salt τ with $\mathcal{G}_2$ in step 2. *Alice* should compute υ after receiving message 2 so that she could compute and pass $\mathcal{G}_0$ with $\mathcal{H}_1$ in step 3. Therefore, *Bob* is able to compute β after receiving message 3. That is, the conventional salt protocol loses the parallel compution of α and β so that the parallel exponentiation cost is to be $4E$ rather than $3E$. However, it is still comparable to other protocols such as AMP$(3E)$, AMP$^+(3E)$, and SRP$(4E)$[1, 9].

3 Conclusion

In this document, we have shown how to avoid the information leakage argument in our previous contribution, "Ultimate Solution to Authentication via Memorable Password"[1],

though we would like to urge it is not critical even in the original AMP. We proposed two extended AMP such as AMP$^+$ and AMP^{++}. Note that $\mathcal{G}_2 = g^{(xe+v)y}$ while the agreed key was still g^{xy} in AMP$^+$. Also note that the agreed key was $g^{(x_1+ex_2)y}$ while $\mathcal{G}_2 = g^{(x_2-v)y}$. Both are clearly preventing the information leakage of the agreed key in $\mathcal{G}_2$ in the way that $\mathcal{G}_2$ and the agreed key are exactly unrelated regading the intractability of the discrete logarithm problem and the Diffie-Hellman problem. The implicit salt scheme discussed in Bellare and Rogaway[2] and the secure prime modulus[3] have been helpful for preserving the efficiency of AMP in AMP^{++} while AMP$^+$ intrinsically preserved the efficiency of AMP.

References

[1] Taekyoung 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/submissions.html#amp*.

[2] 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*.

[3] 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

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

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

[6] 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

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

[8] C.P.Schnorr, "Efficient identification and signatures for smart cards," Crypto 89, LNCS 435, pp.239-252, 1989

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