On Robust Key Agreement Based on Public Key Authentication

Feng Hao
School of Computing Science
Newcastle University, UK
feng.hao@ncl.ac.uk

***Abstract*—This paper discusses public-key authenticated key agreement protocols. First, we critically analyze several authenticated key agreement protocols and uncover various theoretical and practical flaws. In particular, we present two new attacks on the HMQV protocol, which is currently being standardized by IEEE P1363. These attacks suggest the caution one should take when interpreting theoretical results from a formal model. We further point out that many of the protocol failures in the past are caused by sidestepping an important engineering principle, namely "Do not assume that a message you receive has a particular form (such as g^r for known r) unless you can check this". Constructions in the past generally resisted this principle on the grounds of efficiency: checking the knowledge of the exponent is commonly seen as too expensive. In a concrete example, we demonstrate how to effectively integrate the zero-knowledge proof primitive into the protocol design and meanwhile achieve good efficiency. Our new key agreement protocol, YAK, has comparable computational efficiency to the MQV and HMQV protocols with clear advantages on security. Among all the related techniques, our protocol appears to be the simplest so far. We believe simplicity is also an important engineering principle.**

I. INTRODUCTION

In a seminal paper in 1976, Diffie and Hellman started the public key research era by presenting a remarkably simple key agreement protocol based on the intractability of computing discrete logarithm [1]. The protocol works as follows. Suppose two users are Alice and Bob. Let p be a large prime, and α a primitive root modulo p. The original scheme operates in the whole cyclic group Z_p^*. Alice chooses a random value $x \in_R [1, p-1]$ and sends α^x to Bob. Similarly, Bob chooses $y \in_R [1, p-1]$ and sends α^y to Alice. Finally, both parties can compute a common key $K = \alpha^{xy}$.

Later, several changes are made to the original protocol to improve security and efficiency. First, $H(K)$ is used instead of K as the session key where H is a one-way hash function. This is to address the issue that some (least significant) bits of K may be weaker than others [2]. The second change is to move the key agreement operation from the whole group Z_p^* to a large subgroup of prime order q where $q|p-1$. This change is made to address the concern that an active attacker may confine the value K to a small subgroup [24]. However, as we will explain, this does not really solve the problem because the protocol is unauthenticated per se. Finally, it is increasingly popular to implement the Diffie-Hellman protocol using the Elliptic Curve Cryptography (ECC) [10]. Using ECC essentially replaces the underlying (multiplicative) cyclic group with another (additive) cyclic group defined over some elliptic curve. The essence of the protocol remains unchanged.

The acute problem with the Diffie-Hellman key agreement is that it is unauthenticated [2]. While secure against passive attackers, the protocol is inherently vulnerable to active attacks such as the man-in-the-middle attack [6]. This is a serious limitation, which since 1976 has been motivating researchers to find a solution [3]–[5], [7], [9], [11], [13], [20].

To add authentication, we need to start with assuming some shared secret. In general, there are two approaches. The first one assumes Alice and Bob share a symmetric secret: a memorable password. Research following this line is commonly called Password Authenticated Key Exchange (PAKE) [3]–[5]. The second approach assumes Alice and Bob share some asymmetric secret: each party possesses a unique private key and his public key is known by others. In the past literature, protocols under this category are commonly called Authenticated Key Exchange (AKE) [7], [9], [11], [13], [20].

In this paper, we focus on the second category. To better differentiate it from the first category, we will call it Public Key Authenticated Key Exchange (PK-AKE). In the following section, we will review the state-of-the-art in this field.

II. PAST WORK

Before reviewing past techniques in detail, we start by summarizing three general principles. These principles are important because they can help explain most of the protocol failures in the past.

- ***The sixth principle*** – Do not assume that a message you receive has a particular form (such as g^r for known r) unless you can check this [22].
- ***The explicitness principle*** – Robust security is about explicitness; one must be explicit about any properties which can be used to attack a public key primitive, such as multiplicative homomorphism, as well as the usual security properties such as naming, typing, freshness, the starting assumptions and what one is trying to achieve [22].
- ***The extreme-adversary principle*** – Robust security is to protect against an extremely powerful adversary, such that the only powers he does not have are those that would allow him to trivially break any PK-AKE protocol [13].

These principles are simple and intuitive. The first two are time-honored guidance in designing robust cryptographic protocols, defined by Anderson and Needham back in 1995 [22].

The third one is a theoretical principle that we summarize from [13]. This principle is particularly important because it provides the ultimate definition of the "security" of a protocol [13]. In the following review, we will apply this extreme-adversary principle to assess the actual robustness of key exchange protocols.

There is one big family of PK-AKE protocols based on the use of digital signatures [10]. The basic idea is to use a static private key to digitally sign ephemeral public keys (together with some auxiliary inputs such as identities), so that man-in-the-middle attacks can be prevented. It was first described by Diffie, Oorschot and Wiener in the design of the Station-To-Station (STS) protocol [12]. Subsequent signature-based protocols can be seen as variants of the STS protocol.

One well-known member in this family is the SIG-DH protocol due to Canetti and Krawczyk [11]. This protocol is notable for its provable security in a formal model, commonly known as the Canetti-Krawczyk (CK) model. The CK model defines a strong adversary who has the power to corrupt a session and learn all session-specific transient secrets. The goal is that a corrupted session must not impact the security of other sessions.

The SIG-DH protocol is described in Figure 1. It operates in a subgroup of Z_p^* of prime order q. The g is a generator (non-identity element) of the subgroup. The symbols $\hat{A}, \hat{B}$ denote the user identities and g^a, g^b are the users' respective static public keys. The rest symbols are self-explanatory. More details about the SIG-DH protocol can be found in [11].

The provable security of SIG-DH is however disputed by LaMacchia et al [13]. The argument centers on the definition of the "session-specific transient secrets". In [11], the formal proofs only consider the session key and ephemeral exponents as transient secrets. The SIG-DH protocol however does not explicitly specify a digital signature scheme. In fact, for common signature schemes, such as DSA, Schnorr or ElGamal, the signing operation will introduce an additional ephemeral secret (for randomization). If that randomization secret is revealed in a corrupted session, then the static private key will be disclosed. This will surely impact the security of other sessions, thus invalidating the claim in [11]. The same attack generally applies to signature-based PK-AKE protocols.

To address the above deficiency, LaMacchia et al proposed an extended Canetti-Krawczyk (eCK) model [13]. The new model assumes the attacker can learn *all* – instead of parts – of the session specific secrets. Accordingly, the authors presented a NAXOS protocol, and formally proved it secure under the eCK model. Their protocol is shown in Figure 2.

The eCK model claims to be stronger than other formal models such as the CK model [13]. However, this claim is disputed by Cremers [15]. He compares the theoretical properties between the the CK and eCK models, and demonstrates that a protocol proven secure in the eCK model may prove insecure in the CK model and vice versa. In other words, the two models are simply incompatible: neither one is stronger than the other (also see [28]).

The problem in LaMacchia et al's model is that the definition of "session specific secrets" is still ambiguous. Notice in Figure 2, Alice uses $H_1(x, a)$ instead of x on the exponent – a technique known as the "NAXOS trick" [28]. Similarly, Bob uses $H_1(y, b)$ instead of y. The underlying assumption in the NAXOS formal proofs is that the attacker has to steal both the ephemeral secret x and the static private key a in order to learn the exponent. This assumption plays a vital role in proving security in the eCK model. However, one will naturally ask whether $H_1(x, a)$ itself forms part of the "session specific secrets". Allowing a powerful attacker to learn one transient secret x but denying him to learn another transient secret $H_1(x, a)$ contradicts the extreme-adversary principle stated in the NAXOS paper [13].

As stated in [13], there is a secondary reason for using $H_1(x, a)$ instead of x in NAXOX. That is to address the problem that "the random number generator of a party is corrupted". In that case, the ephemeral secret x will have low entropy. Consequently, an attacker may be able to uncover x say by exhaustive search. On the other hand, if the low-entropy x is combined with a high-entropy private key a to form $H_1(x, a)$, the exponent will have high entropy. As plausible as this analysis may sound, it fails to consider the correlation between the exponents. Figure 3 shows a replay attack if the random number generator is corrupted. Assume in one past session, Alice had transferred \$1m to Charlie. Since x has low entropy, with some non-negligible probability the same x value may repeat in a future session. When that occurs, the attacker simply replays the old values Y and M as in the past session, to cause Alice to transfer money again.

This attack shows that hashing x together with a does not really solve any problem. Even worse, it may provide a false sense of security. In fact, if the end user's random number generator is corrupted, no PK-AKE protocols can guarantee security under that setting. The NAXOS protocol is of course no exception.

We now move on to study a different protocol: HMQV (see Figure 4) [7]. The HMQV protocol is modified from MQV [20] with the primary aim for provable security. The modifications come in two flavors. First, HMQV uses a hash function to derive d and e instead of a linear function as defined in MQV. It also mandates the use of a hash function to derive the session key. Second, HMQV provably drops some mandated verification steps in MQV, including the verification of the Proof of Possession (PoP) of the private key during the CA registration and the prime-order validation check of the ephemeral public key.

The changes in the second category are highly controversial despite that they are backed up by a formal model and full proofs [7]. Dropping the public key validations is the direct cause of several attacks against HMQV [14], [17]. In one example, Menezes and Ustaoglu demonstrated a small subgroup confinement attack that could lead to the disclosure of the user's private key [14]. That attack assumes a corrupted session where the attacker can learn the ephemeral exponent. This assumption is allowed in the original adversarial model in HMQV, hence the attack is valid. In the subsequent submission to IEEE P1363 Working Group [8], Krawczyk revised the

	Alice ($\hat{A}, g^a$)		Bob ($\hat{B}, g^b$)
1.	$x \in_R Z_q$	$\xrightarrow{\hat{A}, sid, g^x}$	
2.	Verify signature	$\xleftarrow{\hat{B}, sid, g^y, SIG_B(\hat{B}, sid, g^y, g^x, \hat{A})}$	$y \in_R Z_q$
3.		$\xrightarrow{\hat{A}, sid, SIG_A(\hat{A}, sid, g^x, g^y, \hat{B})}$	Verify signature
	Alice and Bob compute $\kappa = H(g^{xy})$		

Figure 1. SIG-DH protocol. The session identifier *sid* is unique among all sessions owned by $\hat{A}$, the initiator.

	Alice ($\hat{A}, g^a$)		Bob ($\hat{B}, g^b$)
1.	$x \in_R Z_q$	$\xrightarrow{X = g^{H_1(x,a)}}$	Verify X has prime order q
2.	Verify Y has prime order q	$\xleftarrow{Y = g^{H_1(y,b)}}$	$y \in_R Z_q$
	Alice and Bob compute $\kappa = H_2(g^{a \cdot H_1(y,b)}, g^{b \cdot H_1(x,a)}, g^{H_1(y,b) \cdot H_1(x,a)}, \hat{A}, \hat{B})$		

Figure 2. NAXOS protocol. The H_1 and H_2 are two independent hash functions.

	Alice ($\hat{A}, g^a$)		Attacker (pretend "Bob")
1.	x (repeat)	$\xrightarrow{X = g^{H_1(x,a)}}$	Detect same X as in the past
2.	Verify Y has prime order q	$\xleftarrow{Y = g^{H_1(y,b)}}$	Replay old Y
	Pay Charlie \$1m	$\xleftarrow{M = E_\kappa(\text{"Transfer \$1m to Charlie"})}$	Replay old M
	Alice compute $\kappa = H_2(g^{a \cdot H_1(y,b)}, g^{b \cdot H_1(x,a)}, g^{H_1(y,b) \cdot H_1(x,a)}, \hat{A}, \hat{B})$		

Figure 3. Replay attack on NAXOS protocol if x has low entropy

	Alice ($\hat{A}, g^a$)		Bob ($\hat{B}, g^b$)
1.	$x \in_R Z_q$	$\xrightarrow{X = g^x}$	Verify $X \neq 0$
2.	Verify $Y \neq 0$	$\xleftarrow{Y = g^y}$	$y \in_R Z_q$
	$d = \bar{H}(X, \hat{B}),\ e = \bar{H}(Y, \hat{A})$		
	Alice computes: $\kappa = H((YB^e)^{x+da}) = H(g^{(x+da)(y+eb)})$		
	Bob computes: $\kappa = H((XA^d)^{y+eb}) = H(g^{(x+da)(y+eb)})$		

Figure 4. HMQV protocol. $\bar{H}$ and H are two independent hash functions.

HMQV protocol by adding the following check[1]: Alice verifies the term YB^e has the correct prime order and Bob does the same for XA^d. This change prevents the attack reported in [14], but decreases the claimed efficiency of HMQV. The revised HMQV had been included into the IEEE P1363 standards draft (2009-06-30) [29].

However, the revised HMQV still has flaws. First, we present a new "invalid public key attack" that exploits the lax CA requirement in HMQV. In both the original and revised versions of HMQV, CA is only required to check the submitted public key is not 0. The attack works as follows. Assume Bob (attacker) registers a small group element $s \in G_w$ as the public key where $w|p-1$. Bob chooses an arbitrary value $z \in Z_q$. Let $Y = g^z \cdot s'$ where s' is an element in the same small subgroup G_w. Exhaustively, Bob tries every element s' in G_w such that $YB^e = g^z \cdot s' \cdot s^e = g^z$. In other words, the small subgroup elements s and s' cancel each other out. Suppose $\bar{H}$ works like a random oracle as assumed in HMQV (see Figure 4). Then, for each try of s', the probability of finding $s' \cdot s^e = 1$ is $1/w$. It will be almost certain to find such s' after searching all w elements in G_w (if not then change a different z and repeat the procedure). Following the HMQV protocol, Bob sends $Y = g^z \cdot s'$ to Alice. Alice checks YB^e has the correct prime order and computes the session key $\kappa = H((YB^e)^{x+da}) = H(g^{z \cdot (x+da)})$. Because Bob knows z, he can compute the same session key κ and successfully authenticates himself to Alice. In fact, anyone can do the same pre-computation as above and authenticate to Alice as "Bob".

For any PK-AKE protocol, the basic goal of authentication is to assure one party that the other party is the legitimate holder of the supplied public key certificate – more technically, someone who knows the private key [2]. However, in the HMQV case, the private key does not even exist, but the authentication is successful. This indicates a protocol design error.

We now describe a different "wormhole attack" on HMQV. This attack works when the two parties use the same certificate

[1]Actually, Krawczyk does not mandate this check in [8]. He specifies that such a check is necessary to thwart a strong adversary and not necessary if the adversary is less powerful. This is ambiguous. In this paper, we only analyze the stronger (or more secure) version of the revised HMQV, in which the additional check is in place.

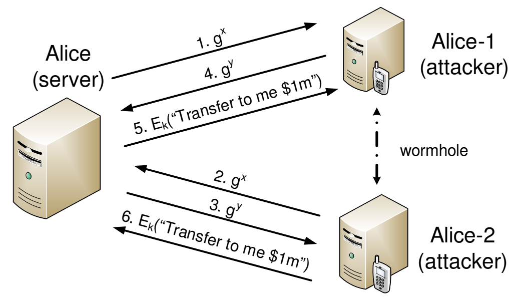

Figure 5. Worm-hole attack on HMQV

for self-communication. Self-communication is considered a useful application in [7]. For example, a mobile user and the desktop computer may hold the same static private key (registering two public key certificates costs more). Krawczyk formally proved that self-communication is "secure" in HMQV [7]. However, the formal model in [7] only considers the user talking to one copy of self, but neglects the possibility that the user may talk to multiple copies of self at the same time.

The following attack works similar to a typical "wormhole attack" in wireless networks where the attacker replicates the identity from one place to another through a wormhole tunnel [6]. Figure 5 illustrates the steps of the attack:

1) Alice initiates the connection to a copy of herself by sending g^x. The connection is intercepted by Mallory who pretends to be Alice-1.
2) Mallory starts a separate session by pretending to be Alice-2. He initiates the connection by sending to Alice g^x (this is possible because HMQV does not require the sender to know the exponent).
3) Alice responds to Alice-2 by sending g^y.
4) Mallory replays g^y to Alice as Alice-1.
5) Alice derives a session key and sends an encrypted message to Alice-1, say: "Transfer to me \$1m".
6) Mallory replays the encrypted message to Alice. (After receiving money from Alice, Mallory disconnects both connections.)

In the above attack, we only demonstrated the attack against the two-pass HMQV (implicit authentication). For the three-pass HMQV (explicit authentication), the attack works exactly the same. Also, we have omitted the identities in the message flows, because they are all identical (see [7]).

This attack is essentially an unknown key sharing attack. Alice thinks she is communicating to a mobile user with the same certificate, but she is actually communicating to herself. The attacker does not hold the private key, but he manages to establish two fully authenticated channels with Alice (server). In a different attacking scenario, if Alice sends an encrypted command "shutdown" to its mobile user in step 5, the same command may be replayed back to Alice to shutdown the system. This shows such an attack can be dangerous. The same attack also applies to other PK-AKE schemes, including NAXOS [13], KEA+ [9], CMQV [18], MQV [20], and SIG-DH [11] etc.

III. The YAK protocol

In this section, we explore a new approach to construct the PK-AKE protocol. So far, almost all of the past PK-AKE protocols [7], [9], [11], [13], [18] have sidestepped the sixth robustness principle that we explained in Section II. The reason has mainly been for the concern on efficiency: verifying the knowledge on the exponent is considered too expensive [7], [11]. In the following sections, we will demonstrate how to effectively integrate the zero-knowledge primitive into the protocol design and meanwhile achieve good efficiency.

Our new PK-AKE protocol is called YAK[2]. For simplicity, we describe it in the DSA-like cyclic group setting [2], [10] (the protocol works basically the same in the ECDSA-like setting where an additive cyclic group over some elliptic curve is used). Let G denote a subgroup of Z_p^* with prime order q in which the Computational Diffie-Hellman problem (CDH) is intractable. Let g be a generator in G (any non-identity element in G can be used as a generator). The two communicating parties, Alice and Bob, both agree on (G, g).

A. stage 1: public key registration

In stage 1, Alice and Bob obtain an authentic copy of each other's static public key. A trivial method is that the two meet in person and exchange public keys. However, this is not scalable. Hence, a Public Key Infrastructure (PKI) is normally needed to distribute authentic public keys: a Certificate Authority (CA) certifies users' public keys and publishes the certificates that anyone can access.

CA Registration: Alice selects a random secret $a \in_R Z_q$ as her private key and sends to the CA g^a with a knowledge proof for a. Similarly, Bob selects $b \in_R Z_q$ as his private key and sends to the CA g^b with a knowledge proof for b.

The CA needs, among several things, to verify the applicant's identity (Distinguished Name) and check the knowledge proof to ensure the Proof of Possession (PoP) of the private key. The PoP check is a mandatory requirement for the CA in PKI standards such as PKCS#10 (see [19]). In practice, the CA may delegate the responsibility of verifying the person's identity to a Registration Authority (RA). The user is normally required to visit the RA in person with a passport or ID card to prove his real identity.

During the registration, the applicant needs to demonstrate the Proof of Possession (PoP) of the private key. There are several ways to do this. One way is to use Zero Knowledge Proof, which is a well-established primitive in cryptography [10]. This technique allows the applicant to prove the knowledge of the exponent without leaking it.

As an example, we can use Schnorr's signature, which is one of the most well-known non-interactive Zero Knowledge Proof techniques. Let H be a publicly-known secure hash function. To prove the knowledge of the exponent for $X = g^x$, one sends {SignerID, OtherInfo, $V = g^v$, $r = v - x \cdot h$} where SignerID is the *unique* user identifier (also called Distinguished Name [2]), OtherInfo includes auxiliary

[2]The yak lives in the Tibetan Plateau where environmental conditions are extremely adverse.

information to indicate this is a request for certifying a static public key and may include other practical information such as the name of the algorithm etc[3], $v \in_R Z_q$ and $h = H(g, V, X, \text{SignerID}, \text{OtherInfo})$. The receiver checks that X lies in the subgroup of prime order q and verifies that $V = g^r X^h$ (computing $g^r X^h$ requires roughly one exponentiation using the simultaneous computation technique [10]). We will assess the cost in more detail in Section VI.

B. stage 2: key agreement

Alice and Bob execute the following protocol to establish a session key. For simplicity of discussion, we explain the case that Alice and Bob have different certificates ($a \neq b$) and will cover self-communication later.

YAK protocol: Alice selects $x \in_R Z_q$ and sends out g^x with a knowledge proof for x. Similarly, Bob selects $y \in Z_q$ and sends out g^y with a knowledge proof for y.

When this round finishes, Alice and Bob verify the received knowledge proof to ensure the other party possesses the ephemeral private key. As explained earlier, we can use Schnorr's signature to realize the knowledge proof. Both parties also need to ensure the identity (i.e., SignerID) in the knowledge proof must match[4] the one in the public key certificate.

Upon successful verification, Alice computes a session key $\kappa = H((g^y \cdot g^b)^{x+a}) = H(g^{(x+a)(y+b)})$. And Bob computes the same key: $\kappa = H((g^x \cdot g^a)^{y+b}) = H(g^{(x+a)(y+b)})$. The protocol has the same round efficiency and symmetric property as the original Diffie-Hellman protocol [1]. Figure 6 shows how to implement the protocol in two passes, as one party usually needs to initiate the connection.

The two-pass YAK protocol can serve as a drop-in replacement for face-to-face key exchange. This is equivalent to Alice and Bob meeting in person and secretly agreeing a common session key. After Alice and Bob depart, they can use the session key to secure the communication. So far, the authentication is implicit: Alice believes only Bob has the same key and vice versa. In some applications, Alice and Bob may want to perform an explicit key confirmation before starting any communication just to make sure the other party actually holds the same session key.

The method for explicit key confirmation is generally applicable to all key exchange protocols. Often, it is considered desirable to use a different key from the session key κ for key confirmation[5], say use $\kappa' = H(K, 1)$. We summarize a few methods here. A simple method is to use a hash function as presented in [3]: Alice sends $H(H(\kappa'))$ to Bob and Bob replies with $H(\kappa')$. Another straightforward way is to use κ' to encrypt a known value (or random challenge) as explained in [2]. Other approaches make use of MAC functions as suggested in [7], [9]. Given that the underlying functions are secure, these methods do not differ significantly in security.

[3]This is similar to the existing practice in OpenSSL that generates a Certificate Signing Request (CSR) for a new DSA key pair. The DSA private key is used to sign the entire request. Verifying the signature is equivalent to verifying the PoP of the private key. Here, we prefer Schnorr's signature, because it is simpler and has well-understood security proofs.

[4]By "match", we mean the identity is identical to the one on the X.509 certificate, or the two have an unambiguous one-to-one mapping relationship. The latter is useful to provide anonymity in key agreement: both parties use pseudo-identities to prove the possession of the ephemeral exponents and they know how to match the pseudo-identities to real ones at the two ends.

[5]Using a different key for key confirmation has a (subtle) theoretical advantage that after the key confirmation, the session key is still indistinguishable from random. However, this trick has limited practical significance and is not used in for example [3], [4].

IV. Security analysis

In this section, we analyze the security of the YAK protocol with rigorous security proofs. Many PK-AKE papers in the past generally adopted the following steps to prove security of a protocol [14].

1) *Security model* – To specify the capabilities of an adversary and how it interacts with honest parties.
2) *Security definition* – To describe the goals of the adversary.
3) *Security proofs* – To prove that breaking the protocol would imply solving some computational problem that is widely believed to be intractable.

However, the practical results have been less than being satisfactory. An outstanding problem lies in how to define a "right" security model in the first place. A right model should specify a *full* set of the attacker's capabilities, hence covering attacking scenarios comprehensively. Although many formal models have been proposed in the PK-AKE field, it is far from clear which exact model is the "right" model [14], [15].

All the existing security models (such as CK, eCK, HMQV etc models) aim to model an adversary in terms of what he is capable of. This methodology evolves progressively over the past two decades by adding more power to the attacker. Still, the resultant models only cover a *subset* of the attacker's capabilities. In consequence, several protocols that have been proven secure in a formal model turn out to be vulnerable to some attacks [14]. In addition, different models are sometimes found incompatible: a protocol proven secure in one model may prove to be insecure in another model and vice versa (see [15]).

Clearly, there is a gap between theory and practice in the specification of a formal model. Bridging the gap has become an active research topic in recent years [15].

In this section, we will attempt a new approach – instead of defining what the attacker is capable of, we focus on what the attacker is *not* capable of. The intuitive idea is to give the fewest possible restrictions on what an attacker is able to do (this idea was originally from LaMacchia et al in [13]). As we will demonstrate, this new approach allows capturing the crux of the extreme-adversary principle more directly.

First, we need to define what are the "session specific secrets" in YAK. Simply put, they include all transient secrets in a session. More specifically, the session specific secrets – for Alice – include the ephemeral exponent x and the raw session key K. This definition has covered the randomization factor v in Schnorr's signature since one can easily compute v from x and r (the r is part of Schnorr's signature). It has also covered the session key κ, which can be computed from $H(K)$. If the attacker is powerful enough to access Alice's session state,

	Alice ($\hat{A}$,g^a)		Bob ($\hat{B}, g^b$)
1.	$x \in_R Z_q$	$\xrightarrow{g^x, KP\{x\}}$	Verify $KP\{x\}$
2.	Verify $KP\{y\}$	$\xleftarrow{g^y, KP\{y\}}$	$y \in_R Z_q$
	Compute $\kappa = H(g^{(x+a)(y+b)})$		$\kappa = H(g^{(x+a)(y+b)})$

Figure 6. YAK protocol

we assume he can learn *all* of the transient secrets including x and K.

In summary, a powerful attacker is able to learn all transient secrets in a session, but he cannot learn the user's private key. Learning the private key would allow the attacker to trivially break any PK-AKE protocol.

First, we formulate the following requirements for the PK-AKE protocol.

1) **Private key security**: An attacker cannot learn the user's static private key even if he is able to reveal all session specific secrets in any session.
2) **Full forward secrecy**: Session keys that were securely established in the past uncorrupted sessions will remain secure in the future even when both users' static private keys are disclosed.
3) **Session key security**: An attacker cannot compute the session key if he impersonates a user but has no access to the user's private key.

These requirements summarize essential security properties of a PK-AKE protocol. They even cover those that are missing in the existing formal model definitions. The first requirement is generally not covered by a formal model, but we think it is crucially important. For example, both the SIG-DH [11] and (original) HMQV [7] protocols have been formally proven secure in the CK model. Yet attacks reported in [13] and [14] show that in both protocols, an attacker is able to disclose the user's private key. In the second requirement[6], we use "full" to distinguish it from the "half" forward secrecy, which only allows one user's private key to be revealed (e.g., KEA+ [9]). In the past literature it is common to add "perfect" before "forward secrecy" [7], [9], [11]. However, we drop "perfect" here because it has no concrete meaning [10], [20]. The third requirement concerns both the secrecy and authenticity of the session key. It has already covered the Key Compromise Impersonation (KCI) attack [20]. The "invalid public key" attack in Section II indicates that HMQV does not satisfy this property.

The strategy of our design is to make the best use of well-established techniques such as Schnorr's signature. This allows us to leverage upon the provable results of Schnorr's signature (see [10], [27]), and thus greatly simplify the security analysis.

First, Let us discuss the private key security. Without loss of generality we assume Alice is honest. Unless mentioned otherwise, this assumption will be made throughout the rest of the analysis. As shown in Figure 7 (1), Mallory totally controls Bob's static and ephemeral private keys; additionally, he has the extreme power that allows him to learn Alice's transient secrets in an arbitrary session. The only power that he does not have is the access to Alice's private key.

Theorem 1 (Private Key Security): An attacker can not learn Alice's static private key even if he is able to learn all transient secrets in any of Alice's sessions.

proof. As shown in Figure 7 (1), an extremely powerful attacker completely corrupted Bob and has access to all of the transient secrets in Alice's session. The knowledge proofs[7] defined in the YAK protocol prove that the attacker knows the values of y and b (i.e., these variables are not correlated with a). He also knows Alice's public key g^a. By revealing Alice's transient secrets in a session, he learns x and the raw session key $K = g^{(a+x)(b+y)}$. But learning K does not give Mallory any information, because he can compute it by himself from $\{x, y, b, g^a\}$. Since Mallory knows the values of $\{x, y, b, g^a\}$, he can effectively simulate the same session all by himself by defining arbitrary values of x, y, b. Clearly, he does not learn any information about Alice's private key from his own simulations.

Intuitively, the above proof assumes an attacker simulating a list of transcripts that include arbitrary values of $\{x,\, y,\, b\}$. By corrupting any of Alice's sessions, the attacker learns nothing more than what he can possibly simulate.

On the other hand, the same simulation does not work in NAXOS and HMQV. Take NAXOS as an example. Assume Bob (the attacker) sends to Alice g^a. By accessing Alice's transient items within the key derivation function, Bob learns $g^{a \cdot a}$. Bob cannot simulate the session because he cannot compute $g^{a \cdot a}$ by himself. In another session, Bob can send to Alice g^{a^2} and then learn g^{a^3} . Similarly, he can learn $g^{a^4}, g^{a^5} \ldots$. In other words, every corrupted session gives the attacker new information that he cannot learn by simulation. The same argument applies to HMQV.

The use of the knowledge proofs greatly simplifies the analysis. Without the knowledge proofs, the simulation in our proof will not work. We illustrate this with an example. Assume there were no knowledge proof (i.e., no PoP check) during the CA registration. Mallory can choose a small subgroup element, e.g., $s \in G_w$ where $w|p-1$. He then registers s/g^y as his static public key. During the key agreement, Alice will compute $K = (s/g^y \times g^y)^{x+a} = s^{x+a}$. If Mallory can also reveal Alice's ephemeral secret x, he can compute $a \bmod w$. This is the same kind of the small subgroup attack as reported against HMQV [14]. Note that in this case, the simulation in the proof no longer works: Mallory does not know the private key b; in fact, the value b does not even exist

[6]It is essentially the same as the weak Perfect Forward Secrecy (wPFS) defined in [7].

[7]If Schnorr's signature is used to realize the knowledge proof, we need to add a random oracle assumption (i.e., a secure one-way hash function), as Schnorr's signature is provably secure in the random oracle model.

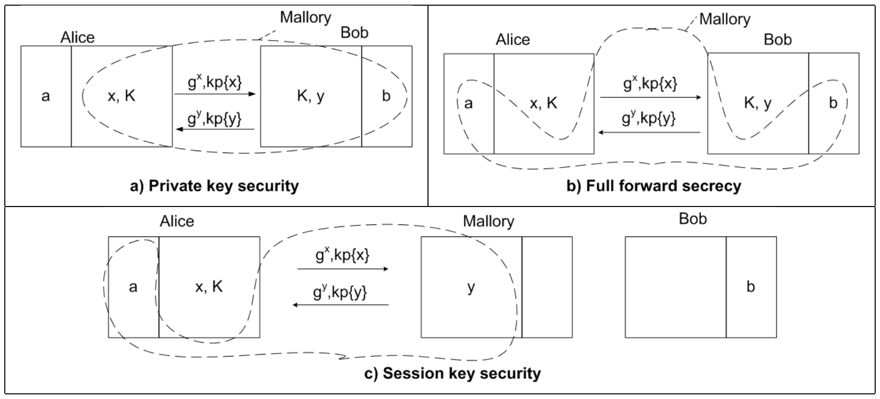

Figure 7. The oracle diagrams for the attacker. Alice is honest.

because the registered public key is a small subgroup element. This example shows the importance of the sixth robustness principle: "Do not assume the message you receive has a particular form (such as g^r for known r) unless you can check this" [22].

Next, we discuss the full forward secrecy requirement. In the definition, we specify that the past sessions must be "uncorrupted"[8], namely the session-specific transient secrets must remain unknown to the attacker. In YAK, this means x, y and K must remain unknown to the attacker. Obviously, knowing K would have trivially broken the past session. Also, if Mallory can learn any ephemeral exponent x or y in the past session in addition to knowing both parties' static private keys (see Figure 7 (2)), he has possessed the power to trivially compromise any PK-AKE. This contradicts the extreme-adversary principle. Therefore, in the following analysis, we assume the attacker knows both Alice and Bob's private keys, but not any transient secrets in the past session.

Theorem 2 (Full Forward Secrecy): Under the Computational Diffie-Hellman (CDH) assumption, an attacker who knows both parties' static private keys but not transient secrets in the past session cannot compute K.

proof. To obtain a contradiction, we assume the attacker can compute $K = g^{(a+x)(b+y)}$. The attacker knows the values of a, b (see Figure 7 (2)). The ephemeral public keys g^x and g^y are public information. Therefore, he can compute g^{ab}, g^{ay} and g^{bx}. Now, we can solve the CDH problem as follows: given g^x and g^y where $x, y \in_R Z_q$, we use the attacker as an oracle to compute $g^{xy} = K/(g^{ab} \cdot g^{ay} \cdot g^{bx})$. This, however, contradicts the CDH assumption.

The above proof shows that the the raw key material K is incomputable to the attacker. In practice, it is not appropriate to directly use the raw K as a session key. A common approach is to apply a key derivation function such as a hash function to produce a key of the desired length, so the session key is $\kappa = H(K)$. This key derivation works exactly the same as in the original Diffie-Hellman protocol [2].

Finally, we study the session key security requirement. As shown in Figure 7 (3), Mallory does not hold Bob's private key but he tries to impersonate Bob. We assume the powerful Mallory even knows Alice's private key a. The only power he does not have is the access to Alice and Bob's session states. If Mallory can access Alice's session state, he can impersonate anyone to Alice – he just needs to "steal" the session key that Alice computes in the transient memory. Similarly, if Mallory can access Bob's session state, he can impersonate Bob to anybody by waiting until Bob computes the session key and then stealing it.

Note in this case, the assumed attacker is less powerful than the one described in Theorem 1. Previously, the attacker was able to corrupt an arbitrary session of Alice's or Bob's. He however had learned no useful information than what he can simulate (see Theorem 1). On discussing the session key security, we assume the attacker no longer has access to either user's session state. This change is necessary, and is consistent with the extreme-adversary principle.

Theorem 3 (Session Key Security): Under the Computational Diffie-Hellman (CDH) assumption, an attacker who impersonates Bob but does not have access to Bob's static private key can not compute K.

proof. The attacker does not possess Bob's static private key, or have access to either Alice or Bob's session state. To obtain a contradiction, we assume Mallory is able to compute $K = g^{(a+x)(b+y)}$. Bob's public key g^b is public information. Mallory knows Alice's private key a. The knowledge proof in the protocol proves that Mallory also knows the value y (see Figure 7 (3)). Hence, he can compute g^{ab}, g^{ay} and g^{xy}. Now, we can solve the CDH problem as follows: given g^b and g^x where $x, b \in_R Z_q$, we use Mallory as an oracle to compute $g^{bx} = K/(g^{ab} \cdot g^{ay} \cdot g^{xy})$. This, however, contradicts the CDH assumption.

Again, the knowledge proofs are essential in the above proof. We use an example to illustrate this. Let us assume there were no knowledge proof required for the ephemeral public key. Now, Mallory can send $Y' = g^{-b}$ to Alice and successfully force the session key to be $\kappa = H(1)$. The removal of the knowledge proof gives the attacker unrestricted freedom to fabricate a message of any form. Note validating

[8]Krawczyk defines a weak Perfect Forward Secrecy (wPFS) and a strong Perfect Forward Secrecy (sPFS) [7]. We observe that both definitions are based on essentially the same assumption: the past sessions were uncorrupted. The only difference is that the latter requires explicit assurance while in the former definition the assurance is implicit. As shown in [7], any two-pass PK-AKE protocol that fulfills wPFS also trivially satisfies sPFS by adding an explicit key confirmation.

the order of Y' does not prevent the attack, because Y' has the correct prime order. Somehow, this example highlights the limitation of the prime-order validation: it only checks whether the message is within the designated group, but fails to check whether it is correlated with other elements in the same group. Using the knowledge proof restricts the attacker's freedom much more stringently, and defeats this attack.

V. Self-communication

The user identity is an important parameter in the protocol definition. In the past literature, almost all PK-AKE protocols readily use the Distinguished Name (DN) in the user's X.509 certificate as the user identity. Unfortunately, the practice of conveniently using DN for the identity carries over to the self-communication mode [7], which caused the "worm-hole" attack in Section II. Self-communication is a special case and should be handled differently.

To enable self-communication in YAK, we need to ensure the SignerID in the Schnorr's signature remains unique. This is to prevent Bob from replaying Alice's signature back to Alice and vice versa. One solution is to simply attach an additional identifier to the mobile stations using the same certificate. For example, when Alice (server) is communicating to the nth copy of herself (mobile station), Alice uses "Alice" as her SignerID to generate the Schnorr's signature and the nth copy uses "Alice-n" as its SignerID. Thus, Alice-n cannot replay Alice's signature back to Alice and vice versa. This solution is also generically applicable to fix the self-communication problem in past protocols [7], [9], [11], [13], [20].

Though self-communication is considered a useful feature [7], one should be careful to enable this feature only when it is really needed. This is because, when enabled, it will have negative impact on the theoretical security. In Section IV, we have explained that, under normal operations (using different certificates), an attacker cannot learn $g^{a \cdot a}$ in any case. However, if self-communication is enabled in YAK, we essentially allow $a = b$, hence the attacker can learn $g^{a \cdot a}$ from a corrupted session. This implies we would need a stronger assumption than CDH to prove the "session key security". This is undesirable, but to our best knowledge, no PK-AKE protocol is reducible to the CDH assumption with the self-communication enabled. In comparison, in NAXOS [13] and HMQV [7], the attacker can learn $g^{a \cdot a}$ from a corrupted session regardless whether the self-communication is enabled (and furthermore he can learn $g^{a^3}, g^{a^4}, \ldots$) .

VI. Comparison

In this section, we compare YAK with past work. First of all, we briefly explain why attacks that we described on past PK-AKE schemes are not applicable to YAK. YAK follows a completely different design from SIG-DH, and is not part of the signature-based PK-AKE family. By design, it prevents the disclosure of session-specific transient secrets from compromising the long-term private key (see Theorem 1). In addition, YAK is free from the two new attacks on HMQV. It resists the invalid public key attack as it requires the Proof of Possession of the private key during the CA registration (following the PKI standards). As explained earlier, it thwarts the worm-hole attack, because it explicitly requires the uniqueness of the user identity even in the self-communication mode.

The resistance of YAK against all these attacks is largely attributed to the use of the Zero Knowledge Proof primitive. Actually, the importance of the ZKP in security protocols has been known for over twenty years, why it has not been commonly applied in key exchange protocols? Many researchers worry that ZKP is too computationally expensive and using it in key exchange will make the protocol too inefficient. But, we argue this worry is not justified, and we believe there is no fundamental conflict between the protocol robustness and efficiency. In the following sections, we will further illustrate this point by comparing YAK with other schemes in detail in terms of security and efficiency.

There are many PK-AKE protocols in the past literature. However, we can only select a few; they include SIG-DH [11], HMQV [7], MQV [20] and NAXOS [13]. These techniques are representative for a comparative analysis. MQV has be widely standardized and applied in practical applications. The rest are all well-known PK-AKE schemes with formal proofs under different formal models. Among them, HMQV is known as the "most efficient" [7] and NAXOS as the "most secure" [13]. Other PK-AKE schemes can be seen as variants of these four.

Table I summarizes the comparison results. The cost is evaluated by counting the number of exponentiations in a DSA-like group setting or the number of multiplications in an ECDSA-like group setting. In the former case, it takes a full exponentiation to validate the prime order of a group element while in latter, this operation is essentially free. This explains the difference of one operation between the second and third columns in Table I.

We also compare the symmetry property of the protocols. The YAK protocol is completely symmetric. The symmetry of a protocol is not any compulsory requirement, but it can significantly simplify the protocol analysis. For example, the proof that Alice's session key is secure must apply to Bob based on symmetry; this reduces the required proofs by half.

The SIG-DH protocol was described in [11] and formally proven secure in the Canetti-Krawczyk (CK) model. However, the paper does not explicitly specify a signature algorithm. This makes it difficult to assert the exact cost and security assumptions because they depend on the choice of the signature algorithm. The attack presented in [13] indicates SIG-DH does not fulfill the private key security requirement. Consequently, it does not satisfy the session key security requirement (since the private key security cannot be assured in the first place). Another limitation with SIG-DH is that if the user's digital signature is captured, his real identity will be revealed.

The HMQV protocol is due to Krawczyk [7]. The protocol is revised in [8] to address the Menezes-Ustaoglu's small subgroup attack by adding a prime-order validation step (otherwise, the protocol will fail the private key security requirement). This revision makes the total number of exponentiations be 3.5. The HMQV only requires the CA to check the submitted public key is not zero. However, as shown in [15], if the attacker is allowed to register "1" as his public

	Exp DL	Mul EC	assum-ptions	CA chk	Pri-key sec	Ses-key sec	FFS	Self com	Allow anonym	Symmetry
SIG-DH	–	–	–	PoP	×	×	✓	×	×	×
HMQV	3.5	2.5	GDH, RO	not $0, 1$	✓	×	✓	×	✓	✓
MQV	3.5	2.5	N/A	PoP	✓	✓	✓	×	✓	×
NAXOS	5 (+1)	4	GDH, RO	not 0	✓	✓	✓	×	✓	×
YAK	5	4	CDH, RO	PoP	✓	✓	✓	✓	✓	✓

Table I
COMPARISON BETWEEN PK-AKE SCHEMES.

key, he can launch an unknown key-sharing attack against the one-pass version of the HMQV protocol. So we add the check that the public key is not "1" either. We need to caution that even so, it is still not sufficient to prevent an unknown key-sharing attack against the HMQV in the post model where the responder's identity is not pre-defined [17]. The "invalid public key" attack shown in Section II indicates HMQV does not fulfill the session key security requirement.

The MQV protocol was first designed by Menezes, Qu and Vanstone [20]. The original MQV design includes the user identities only in the explicit key confirmation stage. Thus, the key confirmation not only serves to confirm the equality of the session key, but also to confirm the identities of the users who are engaged in the key agreement. This arrangement has the drawback that a secure MQV would require 3 passes. As shown by Kaliski, without key confirmation, the 2-pass MQV is subject to an unknown key sharing attack [26]. In [14], Menezes revised the MQV protocol by including the user identities into the key derivation function (similar to HMQV). This change prevents the Kaliski's attack and improves the round efficiency as the 2-pass MQV can now provide implicit authentication. Unfortunately, this change also breaks the symmetry in the original design. Strictly speaking, the modified MQV protocol is no longer one-round (due to the need to determine the order of the two identities in the key derivation function [14]).

The NAXOS protocol is formally proven secure in the extended Canetti-Krawczyk model (eCK) model [13]. The eCK claims to be the "strongest" formal model, but this claim is disputed in [15]. In Section II, we also pointed out a subtle flaw in the definition of the "session specific transient secrets" in the NAXOS security proofs. The NAXOS protocol requires 5 exponentiations (see Figure 2). Same as in HMQV, the protocol allows the CA to certify any non-zero binary strings as public keys. However, NAXOS requires users to verify the other party's certified public key must lie in the correct prime-order group before key agreement. This requires one extra exponentiation, which is not counted in the NAXOS paper. Hence, we add this extra cost (in the braces) in the table.

On the security side, which is our primary concern, the YAK protocol has clear advantages. The security of the protocol (using two different certificates) rests on the Computational Diffie-Hellman (CDH) assumption in the random oracle model. In comparison, the original Diffie-Hellman protocol depends on the same CDH assumption. The random oracle is needed since our protocol depends on the Schnorr's signature. The formal proofs of NAXOS and HMQV depend on a less common Gap Diffie-Hellman (GDH) assumption. The GDH assumes the attacker is unable to solve the CDH problem even when he has access to a Decision Diffie Hellman oracle [7], [13]. Clearly, it is a stronger assumption than CDH.

Finally, we study the efficiency of the protocol. In YAK, Alice needs to perform the following exponentiations: one to compute an ephemeral public key (i.e., g^x), one to compute the knowledge proof for x (i.e., g^{v_x}), two to verify the knowledge proof for y (i.e., Y^q and $g^{r_y}Y^{h_y}$) and finally one to compute the session key $(Y \cdot B)^{x+a}$. Thus, that is five in total: $\{g^x, g^{v_x}, Y^q, g^{r_y}Y^{h_y}, (Y \cdot B)^{x+a}\}$.

Among these operations, some are merely repetitions. To explain this, let the bit length of the exponent be $L = \log_2 q$. Then, computing g^x alone would require roughly $1.5L$ multiplications which include L square operations and $0.5L$ multiplications of the square terms. However, the same square operations need not be repeated for other items with the common base. If we factor this in, it will take $(1+0.5\times 3)L = 2.5L$ to compute $\{g^x, g^{v_x}, g^{r_y}\}$, and another $(1+0.5\times 2)L = 2L$ to compute $\{Y^q, Y^{h_y}\}$ and finally $1.5L$ to compute $(Y \cdot B)^{x+a}$. Hence, that is in total $6L$, which is equivalent to $6L/1.5L = 4$ usual exponentiations. This is quite comparable to the 3.5 exponentiations in MQV (which cannot reuse the square terms since the bases are all different).

VII. CONCLUSION

This paper presents a comprehensive investigation on the subject of the key exchange protocols and is organized in two parts. In the first part, several key exchange protocols are critically analyzed. The analysis reveals a number of practical flaws in the design as well as theoretical deficiencies in the formal model. Two new attacks on the HMQV protocol are also reported; both attacks indicate the basic authentication failure in the protocol despite that HMQV has been "formally proven secure". The reported attacks are caused by sidestepping a prudent engineering principle, namely the sixth principle. In the second part, a new authenticated key agreement protocol, called YAK, is presented. The new protocol follows the sixth principle and is built on well-established cryptographic primitives such as Schnorr signature. It robustly resists all currently known attacks, meanwhile achieving comparable efficiency to the MQV and HMQV protocols.

ACKNOWLEDGMENT

We thank Alfred Menezes and Berkant Ustaoglu for their generous advice and many invaluable comments. We thank anonymous reviewers at FC'10 for many constructive feedbacks on the earlier conference version of the paper. Finally, we are obliged to thank Lihong Yang for helping improve the readability of the paper.

References

[1] W. Diffie and M.E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, pp. 644-654, 1976.

[2] D. Stinson, *Cryptography: theory and practice*, Third Edition, Chapman & Hall/CRC, 2006.

[3] D. Jablon, "Strong password-only authenticated key exchange," *ACM Computer Communications Review*, Vol. 26, No. 5, pp. 5-26, 1996.

[4] S. Bellovin and M. Merritt, "Encrypted Key Exchange: password-based protocols secure against dictionary attacks," Proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.

[5] F. Hao, P. Ryan, "Password authenticated key exchange by juggling," the 16th International Workshop on Security Protocols, SPW'08, Cambridge, UK, May 2008.

[6] R.J. Anderson, *Security Engineering : A Guide to Building Dependable Distributed Systems*, Second Edition, New York, Wiley 2008.

[7] H. Krawczyk, "HMQV: a high-performance secure Diffie-Hellman protocol," Advances in Cryptology – CRYPTO 2005, LNCS 3621, pp. 546-566, 2005. A longer version available at http://eprint.iacr.org/2005/176.pdf.

[8] H. Krawczyk, "HMQV in IEEE P1363," submission to the IEEE P1363 Standardization Working Group, July 7, 2006. Available at http://grouper.ieee.org/groups/1363/P1363-Reaffirm/submissions/krawczyk-hmqv-spec.pdf

[9] K. Lauter, A. Mityagin, "Security analysis of KEA authenticated key exchange protocol," PKC'06, LNCS 3958, pp. 378-394, 2006.

[10] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1996.

[11] R. Canetti , H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," Eurocrypt'01, pp.453-474, 2001.

[12] W. Diffie, P.C. van Oorschot, and M.J. Wiener, "Authentication and authenticated key exchanges," Designs, Codes and Cryptography, pp. 107-125, 1992.

[13] B. LaMacchia, K. Lauter, A. Mityagin, "Stronger security of authenticated key exchange," *Provable Security*, LNCS 4784, pp. 1-16, 2007.

[14] A. Menezes, B. Ustaoglu, "On the importance of public-key validation in the MQV and HMQV key agreement protocols," INDOCRYPT'06, LNCS 4329, pp. 133-147, 2006.

[15] C.J.F. Cremers, "Session-state reveal is stronger than ephemeral key reveal: attacking the NAXOS authenticated key exchange protocol," ACNS'09, LNCS 5536, pp. 20-33, 2009.

[16] F. Bao, R.H. Deng, H. Zhu, "Variations of Diffie-Hellman problem," Proceeding of Information and Communication Security, LNCS 2836, pp. 301-312, 2003.

[17] A. Menezes, B. Ustaoglu, "Comparing the pre- and post-specified peer models for key agreement," Information Security and Privacy, LNCS 5107, pp. 53-68, 2008.

[18] B. Ustaoglu, "Obtaining a secure and efficient key agreement protocol for (H)MQV and NAXOS," *Designs, Codes and Cryptography*, Vol. 46, No. 3, pp. 329-342, 2008.

[19] C. Mitchell, *Security for Mobility*, The Institution of Electrical Engineers, 2004.

[20] L. Law , A. Menezes , M. Qu , J. Solinas , S. Vanstone, "An efficient protocol for authenticated key agreement," *Designs, Codes and Cryptography*, Vol. 28 , No. 2, pp. 119-134, 2003.

[21] C. Boyd, A. Mathuria, *Protocols for authentication and key establishment*, Springer-Verlag, 2003.

[22] R.J. Anderson, R. Needham, "Robustness principles for public key protocols," Crypto'95, LNCS 963, pp. 236-247, 1995.

[23] O. Goldreich, S. Micali and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," Proceedings of the nineteenth annual ACM Conference on Theory of Computing, pp. 218-229, 1987.

[24] C.H. Lim and P.J. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup," Crypto '97, LNCS 1295, pp. 249-263, 1997.

[25] M. Bellare, R. Canetti, H. Krawczyk, "A modular approach to the design and analysis of authentication and key exchange protocols," Proceedings of the thirtieth annual ACM symposium on Theory of Computing, pp. 419-428, 1998.

[26] B. Kaliski, "An unknown key-sharing attack on the MQV key agreement protocol," *ACM Transactions on Information and System Security,* Vol. 4. No. 3, 2001, pp. 275–288.

[27] C.P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, Vol. 4, No. 3, pp. 161-174, 1991.

[28] B. Ustaoglu, "Comparing SessionStateReveal and EphemeralKeyReveal for Diffie-Hellman protocols," The Provable Security Conference, ProvSec'09, LNCS, Nov, 2009.

[29] IEEE P1363 Standard Specifications For Public-Key Cryptography, http://grouper.ieee.org/groups/1363/index.html.