Ace: An Efficient Key-Exchange Protocol for Onion Routing

Michael Backes
Saarland University and MPI-SWS
Saarbrücken, Germany
backes@cs.uni-saarland.de

Aniket Kate
MMCI, Saarland University
Saarbrücken, Germany
aniket@mmci.uni-saarland.de

Esfandiar Mohammadi
Saarland University
Saarbrücken, Germany
mohammadi@cs.uni-saarland.de

ABSTRACT

The onion routing (OR) network Tor provides privacy to Internet users by facilitating anonymous web browsing. It achieves anonymity by routing encrypted traffic across a few routers, where the required encryption keys are established using a key exchange protocol. Goldberg, Stebila and Ustaoglu recently characterized the security and privacy properties required by the key exchange protocol used in the OR network. They defined the concept of one-way authenticated key exchange (1W-AKE) and presented a provably secure 1W-AKE protocol called ntor, which is under consideration for deployment in Tor.

In this paper, we present a novel 1W-AKE protocol Ace that improves on the computation costs of ntor: in numbers, the client has an efficiency improvement of 46% and the server of nearly 19%. As far as communication costs are concerned, our protocol requires a client to send one additional group element to a server, compared to the ntor protocol. However, an additional group element easily fits into the 512 bytes fix-sized Tor packets (or cell) in the elliptic curve cryptography (ECC) setting. Consequently, our protocol does not produce a communication overhead in the Tor protocol. Moreover, we prove that our protocol Ace constitutes a 1W-AKE. Given that the ECC setting is under consideration for the Tor system, the improved computational efficiency, and the proven security properties make our 1W-AKE an ideal candidate for use in the Tor protocol.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; D.2.8 [**Software**]: Security and Protection—*Cryptographic controls*; E.3 [**Data**]: Data Encryption—*Public key cryptosystems*

Keywords

Tor, Onion Routing, Circuit Construction, Authenticated Key Agreement, One-way Anonymity

WPES'12, October 15, 2012, Raleigh, North Carolina, USA.

1. INTRODUCTION

The onion routing (OR) network Tor [18] has been immensely successful as a privacy enhancing technology. It currently employs nearly three thousand dedicated routers (or OR nodes) and serves hundreds of thousands of users all over the world. With its recently observed utility as a censorship-resistant tool these numbers are bound to grow swiftly.

While utilizing the Tor network to access the Internet anonymously, a user constructs a *circuit* choosing a small ordered subset of (usually three) OR nodes, such that the chosen nodes route the user's traffic over the path formed. For the anonymity in onion routing, it is important that an OR node should not be able to determine the circuit nodes other than its predecessor and successor, while routing the user's messages. In the OR protocol, the user achieves this property by sending her every message in the form of an *onion*—a message wrapped in multiple layers of symmetric-key encryption (one layer per selected node). The symmetric keys are agreed upon during an initial *circuit construction* phase using a public-key infrastructure (PKI) implemented using a small set of *directory servers* that also provide routing information for the OR nodes to the users. The key cryptographic challenges in the OR protocol are to securely agree upon the symmetric keys, and then to use those to achieve confidentiality and integrity [2]. In this work, we concentrate on the first challenge.

Tor currently uses an interactive forward-secret key-exchange protocol called the Tor authentication protocol (TAP) in a *telescoping (or multi-pass)* fashion to agree upon the required symmetric keys [6]. However, with its atypical use of an RSA encrypted group element (or pseudonym), TAP is considered to be inefficient. Øverlier and Syverson [15] suggested an efficient replacement for TAP (their fourth protocol) using a half-certified Diffie-Hellman (DH) key agreement [12, §12.6]. Recently Goldberg, Stebila and Ustaoglu showed an attack on the fourth protocol in [15] that allows an adversary to impersonate an honest server (a router in Tor) to an honest client (a user in Tor) [7]. They also defined the concept of one-way authenticated key exchange (1W-AKE), fixed the fourth protocol [15] to obtain a provably secure construction called the ntor protocol, and described its utility towards onion routing. However, while obtaining a provably secure construction, they sacrificed computational efficiency to a certain extend. In particular, every ntor instance requires two *online* discrete logarithmic (DLog) exponentiations on the client side and 1.33 exponentiations on the server side in ntor, where only one online exponentiation

each was required on the both sides in the original fourth protocol in [15]. In this paper, we work towards a computationally more efficient 1W-AKE protocol using a practical concession provided by the Tor protocol.

Contributions.

We present a novel 1W-AKE protocol Ace (anonymous circuit establishment) that achieves an efficiency improvement of 46% at the client-side and of nearly 19% at the server-side, compared to the ntor protocol. The crux is to use as a pseudonym two randomly chosen group elements on the client side instead of one. In this way, we are able to use Shamir's multi-exponentiation trick on both the server and the client side, requiring only 1.17 online exponentiations for the key-exchange. These requirements can be further dropped to 1.08 exponentiations in the elliptic curve cryptographic (ECC) setting as it provides DLog group inverses for free.

Our requirement of sending two group elements from the client to the server may look an impeding factor in terms of communication. However, thanks to the fixed sized packets (or cells) of size 512 bytes in Tor, two group elements of size 32 bytes each in the ECC setting can easily be accommodated in a single cell. Given that the ECC setting is under consideration for the Tor protocol [11], our protocol does not affect the practical communication time of Tor circuit construction at all. We also prove Ace secure using the definition for 1W-AKE that has been introduced by Goldberg, Stebila, and Ustaoglu [7].

Outline.

Section 2 discusses the previous work on 1W-AKE. Section 3 introduces the Ace protocol. Section 4 compares the computational efficiency and the message sizes of the Ace protocol with the previous protocols. Section 5 reviews the security requirements for a 1W-AKE protocol, and shows that Ace indeed constitutes a 1W-AKE protocol. Section 6 concludes and discusses some topics for the future work.

2. BACKGROUND

This section discusses previous work on 1W-AKE protocols. Section 2.1 present the current Tor Authentication protocol (TAP). Section 2.2 discusses a one-way authentication protocol by Shoup that enriches the DH key exchange with a public-key signature. Section 2.3 illustrates the ØS protocol by Øverlier and Syverson, which is efficient but insecure. Section 2.4 presents the ntor protocol, which fixes the issues of the ØS protocol but is less efficient. In the end, Section 2.5 briefly discusses why we do not consider non-interactive key exchange methods.

A comparative overview of these four key exchange protocols and our protocol Ace is presented in Figure 5.

2.1 The current Tor Authentication Protocol

The current Tor authentication protocol (TAP) basically performs a DH key-exchange where the authentication of the server is ensured by encrypting the first DH message g^x under the public key of the server, for a generator g. Using public-key encryption, however, it is inefficient; therefore, a more efficient key exchange is desirable.

2.2 The A-DHKE Protocol

Shoup presented a 1W-AKE protocol A-DHKE that relies on public-key signatures [16] and proved A-DHKE secure[1]. In A-DHKE basically, the DH key exchange is enriched in the second message with a signature of the server on the ephemeral key g^x of the client and the ephemeral g^y of the server. The key derivation function is computed as in a DH key exchange. This protocol only needs 1 online exponentiation as in the usual DH key exchange, but it additionally requires the protocol to compute 1 online signature. Therefore, the efficiency of A-DHKE depends upon the efficiency of the signature scheme.

2.3 The ØS Protocol

Øverlier and Syverson proposed a series of more efficient key-exchange protocols for future deployment in Tor, culminating in their fourth protocol [15]. This fourth protocol basically enhances the DH protocol with a long-term key g^b of the server. Neglecting the session id and the key-confirmation message, the protocol works as follows. The client sends a fresh ephemeral key g^x to the server. The server draws a fresh ephemeral key g^y, computes the session key $(g^x)^{b+y} = g^{x(b+y)}$, and sends g^y back to the client, which compute $(g^b g^y)^x = g^{(b+y)x}$.

An Attack on the ØS Protocol.

Unfortunately, there is a man-in-the-middle attack against this protocol [7]. The attacker intercepts the initial message g^x, draws a fresh g^y, and responds with $g^y/g^b = g^{y-b}$, where g^b is the public key of the server. Then, the client computes the session key $(g^b g^{y-b})^x = g^{yx}$ and the attacker computes $(g^x)^y$. Figure 5 in the appendix illustrates this attack.

2.4 The ntor Protocol

Goldberg, Stebila, and Ustaoglu [7] present a fixed version of the ØS protocol, the ntor protocol. Moreover, the authors proved that ntor is 1W-AKE secure (see Section 5.1).

A closer look at the session key g^{xy+xb} in the ØS protocol reveals that for fixing the protocol it suffices to separate the term xy from the term xb. In ntor, this separation is achieved by applying a hash function H to these terms: $\mathsf{H}(g^{xy}, g^{xb})$.[2] Neglecting the session keys and the key confirmation message, in ntor the client sends a fresh ephemeral key g^x to the server, The server draws a fresh ephemeral key g^y, computes the session key as $\mathsf{H}((g^x)^y, (g^x)^b)$, and responds with g^y to the client.

The security of ntor is bought at a price of efficiency: the client has to compute 2 full exponentiations, and the server has to compute 1.33 exponentiations, using square-and-multiply optimizations since the base of $(g^x)^y$ and $(g^x)^b$ is the same.

2.5 A Note on Non-Interactive KE

In contrast to the presented interactive key-exchange a single-pass construction using a non-interactive key exchange is possible as well. However, achieving forward secrecy of the user's circuits without regularly rotating the PKI keys for all Tor nodes is not possible [9], and the periodic public key

[1]Technically, Shoup proved A-DHKE secure in another model, but it is easy to see that A-DHKE also satisfies the 1W-AKE definition.

[2]g^{xy}, g^{xb} denotes the concatenation of g^{xy} and g^{xb}.

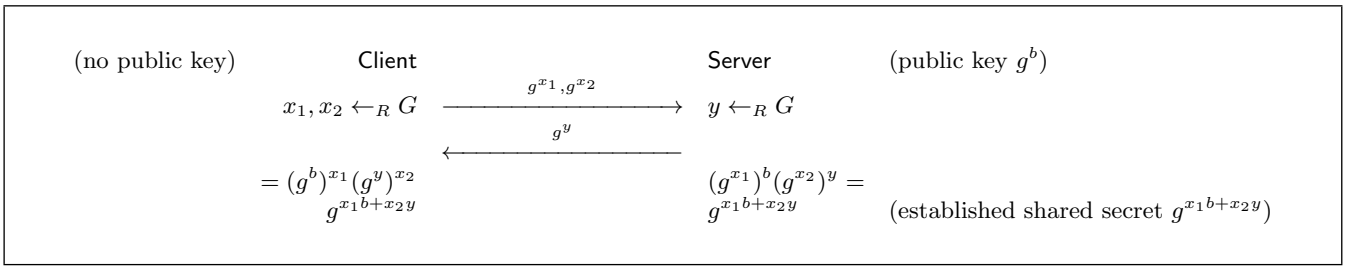

Figure 1: An overview of Ace: G is the exponent group.

rotation should be avoided for scalability reasons. There has also been attempts to solve this problem by introducing the identity-based setting [8, 9] or the certificate-less cryptography setting [5]. However, the key authorities required in these constructions can be difficult to implement in practice. (For a detailed discussion, we refer to [2, Sec. 2.1].)

3. THE Ace PROCOTOL

Along the lines of the ØS protocol and the ntor protocol, the Ace protocol constitutes a one-way authenticated variant of the DH key exchange. The crux of Ace is that it trades computational efficiency with communication efficiency. However, we aim at improving the key-exchange in the Tor protocol, and it turns out that in the ECC setting a large fragment of the Tor packets are actually unused during the key exchange. As the ECC setting is currently under consideration for deployment in Tor, using Ace will not produce a communication overhead for the Tor protocol's key-exchange yet gaining efficiency in terms of computation.

Notation.

We often denote the long-term secret key of a party A as a and correspondingly g^a as the public key, for a given generator g. In the security analysis and the detailed presented of Ace, we also talk about A as an identifier. At these place, we denote the long-term public key of a party A as pk_A and the corresponding secret key as sk_A.

For assigning the result of a (possibly randomized) computation A to a variable v, we write $v \leftarrow A$. Similarly, we write $v \leftarrow V$ to denote the assignation of a value V to the variable v. Moreover, we write $v \leftarrow_R S$ for denoting that a uniformly chosen element from a set S is assigned to the variable v. For the security parameter we use the letter η. In this work, we only consider probabilistic polynomial-time bounded machines, denoted as *ppt machines*. In abuse of notation, we also call a randomized algorithm that is polynomially bounded a *ppt algorithm*. We denote the hash of the concatenation of several values $v_1, \ldots, v_2$ as $\mathsf{H}(v_1, \ldots, v_2)$.

3.1 The Construction

Recall that in the ØS protocol the client sends an ephemeral key g^x and the server responds with an ephemeral key g^y, resulting in a session key g^{xb+xy} (g^b being the server's certified long-term key). The main problem in ØS is that the two terms xy and xb are not separated, allowing the attacker to choose $y := y' - b$, by computing $g^y := g^{y'}/g^b$, and impersonating any server. In the ntor protocol this separation is achieved by letting the session key be the hash of g^{xb} and g^{xy}. This remedy comes at the price of a loss in efficiency. In Ace we achieve this separation by letting the ephemeral key be a pair (g^{x_1}, g^{x_2}): the session key is then $g^{x_1 b + x_2 y}$.

In Ace the client sends an ephemeral pair (g^{x_1}, g^{x_2}) and the server responds an ephemeral key g^y. Then, the client computes the session key as $(g^b)^{x_1}(g^y)^{x_2} = g^{bx_1 + yx_2}$ and the server as $(g^{x_1})^b(g^{x_2})^y = g^{x_1 b + x_2 y}$. Figure 1 gives an overview of the Ace protocol.

The Protocol in Detail.

Figure 2 presents the Ace protocol in detail by showing the pseudo code for the initialization algorithm Init, the response algorithm Resp, and the final key computation algorithm CompKey. Let (pk_Q, sk_Q) be the static key pair for Q. We assume that P holds Q's certificate (Q, pk_Q). Recall that cs is the queue of already chosen ephemeral key pairs (x, g^x) of which g^x is already leaked to the attacker.

The algorithm Init is called for initiating a new session. It expects as input the server's identity, two strings new_session, Ace, and the queue cs. Then, either a fresh ephemeral keys is chosen or an already chosen key is popped from the queue cs. Thereafter, the local session identifier is set by applying the ephemeral pair (x_1, x_2) to a collision resistant hash function H_{st}.[3] Then, the session information of the client is stored in the variable $st(\Psi)$, and the client's ephemeral keys (g^{x_1}, g^{x_2}) together with the server identity Q and a string Ace is output as a network message m along with the session's state $st(\Psi)$ and the session identifier Ψ.

The algorithm Resp is called by the server Q for responding to a session initialization. Resp expects as input the secret key sk_Q of Q, Q's identity, the network message (Ace, X_1, X_2) from the initialization, and the queue cs. First, again a ephemeral key (y, g^y) is chosen, either freshly or from cs. Then, it is verified that X_1, X_2 are in the public key group G^*_η. Thereafter, the local session identifier Ψ_Q is computed by applying the hash function H_{st} to the ephemeral key g^y. Thereafter, the two keys k_m, k_s are derived from $g^{x_1 sk_Q + x_2 y}$ by applying the hash function H to $g^{x_1 sk_Q + x_2 y}$, the session information X_1, X_2, Y, pk_Q, and the protocol name Ace. Since we need k_m and k_s for the proof to be independent, we consider H is a random oracle in the proof. k_m is used for the key-confirmation message, and k_s is the resulting session key. Then, the key confirmation message is computed by basically applying a Mac to all public information using the key k_m. Thereafter, we output the session identifier, the session information, and as a network message the Mac tag, the ephemeral key g^y and a string Ace.

The algorithm CompKey is called by the client for completing the key-exchange. CompKey expects the public key of the server Q, the network message (Ace, Y, t_Q) from Q, and the temporary session state $(Q, (x_1, x_2), (g^{x_1}, g^{x_2}))$. First,

[3]The purpose of this hash function is merely to reduce the space of the session key; therefore, we only need to require collision resistance.

$\mathsf{Init}(Q, (\eta, \mathsf{new\_session}, \mathsf{Ace}), cs)$:
  if $cs = \emptyset$ **then**
    $x_1, x_2 \leftarrow_R G_\eta$
    compute g^{x_1},g^{x_2}
  else
    $(x_1, g^{x_1}), (x_2, g^{x_2}) \leftarrow \mathsf{pop}(cs)$
  set session id $\Psi \leftarrow \mathsf{H}_{st}(g^{x_1}, g^{x_2})$
  set $st(\Psi) \leftarrow (Q, (x_1, x_2), (g^{x_1}, g^{x_2}))$
  set $m \leftarrow (\mathsf{Ace}, Q, (g^{x_1}, g^{x_2}))$
  output $(m, st(\Psi), \Psi)$

$\mathsf{Resp}(sk_Q, Q, (\eta, \mathsf{Ace}, X_1, X_2), cs)$:
  if $cs = \emptyset$ **then** $y \leftarrow_R G_\eta$ **else** $y \leftarrow \mathsf{pop}(cs)$
  verify that $X_1, X_2 \in G_\eta^*$
  set session id $\Psi_Q \leftarrow \mathsf{H}_{st}(g^y)$
  compute $(k_m, k_s) \leftarrow \mathsf{H}(X_1^{sk_Q} \cdot X_2^y, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$
  compute $t_Q \leftarrow \mathsf{Mac}(k_m, (Q, g^y, X_1, X_2, \mathsf{Ace}, \mathsf{server}))$
  set $m_Q \leftarrow (\mathsf{Ace}, g^y, t_Q)$
  $out \leftarrow (k_s, \star, (X_1, X_2), (g^y, g^{sk_Q}))$
  output (m_Q, out, Ψ_Q)

$\mathsf{CompKey}(pk_Q, (\eta, \mathsf{Ace}, Y, t_Q), \Psi, (Q, (x_1, x_2), (g^{x_1}, g^{x_2})))$:
  verify that $Y \in G_\eta^*$
  compute $(k_m, k_s) \leftarrow \mathsf{H}(pk_Q^{x_1} \cdot Y^{x_2}, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$
  if $\mathsf{Mac}(k_m, (Q, Y, g^{x_1}, g^{x_2}, \mathsf{Ace}, \mathsf{server})) = t_Q$ **then**
    $out \leftarrow (k_s, Q, (g^{x_1}, g^{x_2}), (Y, pk_Q))$
    output out

If any verification fails, the party erases all session-specific information and aborts the session.

Figure 2: The Ace protocol: G_η is the group of the secret keys, G_η^* is the group of the public keys. $\mathsf{Gen}(1^\eta)$ outputs a pair (x, g^x) for a random element x of G_η.

we verify that Y is indeed a group element of G_η^*. Then, we compute the key confirmation key k_m and the session key k_s similar as in Resp, and verify the key confirmation message t_Q from the server with k_m. Thereafter, we output the session information $(k_s, Q, (X_1, X_2), (Y, pk_Q))$.

4. PERFORMANCE COMPARISON

In this section, we compare the performance of the Ace protocol with the relevant key agreement protocols.

We consider $\eta = 128$-bit security and use the elliptic curve cryptographic (ECC) setting with points (compressed form) of size $p = 256$ bits, such as provided by Dan Bernstein's Curve25519 [3]. For the finite field setting ($\mathbb{F}$), we consider a DH modulus of size just $p = 2048$ bits to model 128-bit security. In these setting, we compare computational efficiency and message sizes of our protocol with the TAP protocol, the A-DKHE protocol [16], the fourth protocol by Øverlier and Syverson, the multi-pass pairing-based onion routing (PB-OR) protocol [9] and the ntor protocol.

4.1 Computational Efficiency

Table 1 compares computational efficiency and security of the above mentioned relevant key exchange schemes. We also include the unauthenticated and insecure Diffie-Hellman (DH) key exchange protocol to set the baseline for the required computation, where one (online) exponentiation is enough on both client and server sides. The TAP protocol also requires one exponentiation on both sides; however, it requires one RSA encryption on the client side and one RSA decryption on the server side, and the latter operation increases the server-side computational cost significantly.

The fourth protocol by Øverlier and Syverson is although as efficient as the unauthenticated DH key exchange, it is insecure. The ntor protocol requires two exponentiations on both client and server sides; however, the two exponentiations on the server side use the same base, and they can be parallelized [13] to reduce the computational cost to 1.33 exponentiations for $\eta = 128$. Although our Ace protocol naïvely also requires two exponentiations on both client and server sides, exponentiations on both sides are actually multi-exponentiations and using Shamir's trick [12, Algo. 14.88] can be reduced to only 1.17 exponentiations on both sides for $\eta = 128$. In the ECC setting, where group inverses come for free, the number of exponentiations can be further reduced to 1.08 exponentiations using Avanzi's algorithm [1] based on a sliding windows method for the joint sparse form [17].

The A-DHKE protocol uses one signature generation on the server side and one signature verification on the client side alone with one exponentiation on each side for the session-key computation, and its efficiency depends upon the efficiency of the signature scheme used. In our ECC Curve25519 setting, the signature generation is expensive and A-DHKE is significantly inefficient than the Ace protocol. However, Bernstein et al. find that high-speed signatures are possible using table lookups and a twisted Edwards curve [4]. Using this signature scheme, server-side computation for A-DHKE may become nearly equal to a single exponentiation. (See the discussion on the tor-dev mailing list [10].) Nevertheless, we observe that the multi-exponentiation techniques used in the Ace protocol can also benefit from table lookups; hence, the performance of Ace protocol will remain comparable to A-DHKE over the twisted Edwards curve.

We also include the multi-pass PB-OR protocol in our comparison for completeness. It asks for a bilinear pairing along with an exponentiation on both sides. However, the protocol belongs to the identity-based setting and the capability of the Tor network to implement the required setup assumptions is not clear.

4.2 Message Sizes

All of the above discussed relevant key exchange protocols except our Ace protocol require one group element to be communicated from the client to the server.[4] For $\eta = 128$, this asks for 256 bytes in the finite field setting, and 32 bytes in the ECC setting. In our Ace protocol, the client communicates two group elements to the server. In the finite field setting, this asks for 512 bytes in the finite field setting, and 64 bytes in the ECC setting.

However, the Tor uses cells of size 512 bytes, and in the ECC setting, sending 64 bytes instead of 32 bytes does not affect the Tor protocol. As the ECC setting is under con-

[4]Note that the client communicates an RSA-encrypted group element in TAP.

Table 1: Comparison between computational cost of relevant key exchange schemes for 128-bit security

Protocol	Exponentiations (client)		Exponentiations (server)		Security
	Off-line	On-line	Off-line	On-line	
DH	1	1	1	1	insecure
A-DHKE [16]	1	$1+1^{*}$	1	$1+1^{*}$	secure
TAP [6]	1	$1+1^{\dagger}$	1	$1+1^{\dagger}$	secure
ØS [15]	1	1	1	1	insecure
Multi-pass PB-OR [9]	1	$1+1^{\ddagger}$	1	$1+1^{\ddagger}$	secure
ntor [7]	1	2	1	1.33	tight
Ace **(this paper)**	2	1.08(1.17)	1	1.08(1.17)	tight

* A-DHKE requires a signature generation on the server side and a signature verification on the client side.
† TAP requires an RSA encryption on the client side and an RSA decryption on the server side.
‡ Multi-pass PB-OR requires a bilinear pairing on both client and server sides.

upon $\mathsf{send}^P(params, Q)$**:**
$(m, st, \Psi) \leftarrow \mathsf{Init}(Q, params, cs)$
$akest^P(\Psi) \leftarrow (Q, st)$; send (m, Ψ)

upon $\mathsf{send}^P(\Psi, m)$ **and** $akest^P(\Psi) = \perp$**:**
$(m', (k, \star, st), \Psi) \leftarrow \mathsf{Resp}(sk_P, P, m, cs)$
$resust^P(\Psi) \leftarrow (k, \star, st)$; send m'

upon $\mathsf{send}^P(\Psi, m)$ **and** $akest^P(\Psi) \neq \perp$**:**
$(Q, st) \leftarrow akest^P(\Psi)$; check for a valid pk_Q
$(k, Q, st) \leftarrow \mathsf{CompKey}(pk_Q, m, \Psi, (Q, st))$
erase $akest^P(\Psi)$; $resust^P(\Psi) \leftarrow (k, Q, st)$

upon $\mathsf{reveal\_next}^P$**:**
$(x, X) \leftarrow \mathsf{Gen}(1^\eta)$; append (x, X) to cs; send X

upon $\mathsf{partner}^P(X)$**:**
if a key pair (x, X) is in the memory **then** send x

upon $\mathsf{sk\_reveal}^P(\Psi)$**:**
if $resust^P(\Psi) = (k, Q, st)$ **then** send k

upon $\mathsf{establish\_certificate}(Q, pk_Q)$**:**
register the public key pk_Q for the party Q

upon $\mathsf{test}^P(\Psi)$**:** *(one time query)*
$(k, Q, st) \leftarrow resust^P(\Psi)$
if $k \neq \perp$ and $Q \neq \star$ and Ψ is 1W-AKE fresh **then**
if $b = 1$ **then** send k **else** send $k' \leftarrow_R \{0,1\}^{|k|}$

for every query (consistency check)
if in a client session only one key X_i is partnered **then** send X_{3-i}

Figure 3: 1W-AKE Security Challenger: $\mathsf{Ch}_b^{\mathrm{KE}}(1^\eta)$. If any invocation outputs $\perp$, the challenger erases all session-specific information for that session and aborts that session.

sideration for the Tor protocol [11], we find our protocol to be more aptly suited to replace TAP instead of ntor.

5. SECURITY ANALYSIS

This sections presents a security analysis of Ace. First, Section 5.1 reviews the security requirements for 1W-AKE protocol. Second, Section 5.2 discusses the security of Ace.

5.1 Security Definition of Anonymous 1W-AKE

Goldberg, Stebila, and Ustaoglu [7] formalize the security of a 1W-AKE protocol between an anonymous client and an authenticated server by requiring the following three properties. First, the protocol should produce correct results if both parties are honest (*correctness*). Second, even a malicious attacker that can compromise single sessions and introduce fake identities cannot learn anything about the session key of uncompromised sessions (*1W-AKE security*). In particular, 1W-AKE security implies that the attacker cannot impersonate a server. Third, a server should not be able to see any difference while communicating with two different clients (*1W-anonymity*). In this section, we review the three notions Correctness, 1W-AKE security, and 1W-anonymity.

A 1W-AKE protocol is a tuple of ppt algorithms $\mathsf{AKE} = (\mathsf{Gen}, \mathsf{Init}, \mathsf{Resp}, \mathsf{CompKey})$. Gen is called for generating temporary asymmetric keys, Init is called at the client for starting a 1W-AKE, Resp is called at the server for responding to a 1W-AKE initialization, and CompKey is again called at the client for verifying the key confirmation message and computing the key. We assume that every party $P \in \{P_1, \ldots, P_n\}$ can register public keys, and every party can obtain certificates for other parties' public keys and verify them.

Correctness of 1W-AKE.

Correctness states that if all parties behave honestly, the protocol succeeds. Also, the correctness property requires the 1W-AKE algorithms to finally output a vector $\vec{v} = (v_1, v_2)$ that contains all ephemeral information and the long-term public key. For Ace, $\vec{v} = ((g^{x_1}, g^{x_2}), (g^y, g^{sk_P}))$, where x_1, x_2 are the ephemeral secret keys of the client for that session, y is the ephemeral secret key of the server $P \in \{P_1, \ldots, P_n\}$ for that session, and sk_P is the secret long-term key of the server. Moreover, the 1W-AKE algorithms output the session key k_s, and the ID of the peer party, where the client outputs the actual ID of the server, and the server only outputs $\star$, since the client is anonymous.

Definition 1 (Correctness of 1W-AKE). *Let a PKI be given, i.e., for every party $P \in \{P_1, \ldots, P_n\}$ every party knows a (certified) public key pk_P and P itself also knows*

the corresponding secret key sk_P. Let AKE := (Gen, Init, Resp, CompKey) *be a tuple of polynomial-time bounded randomized algorithms. We say that* AKE *is a* correct one-way authenticated key exchange *protocol if the following holds for all parties A, B:*

$$\Pr\big[(m, st, \Psi) \leftarrow \mathsf{Init}(Q, m, cs),\ (m', (k, \star, \vec{v}), \Psi_Q) \leftarrow \mathsf{Resp}(sk_Q, Q, m, cs),\ (k', Q, \vec{v}') \leftarrow \mathsf{CompKey}(pk_Q, m', \Psi, st) : k = k' \text{ and } \vec{v} = \vec{v}'\big] = 1.$$

1W-AKE Security.

We require that the attacker does not learn anything about the key and is not able to impersonate honest parties. This notion is formalized by requiring that even in the presence of an attacker that can send commands to each party, establish several concurrent sessions, compromise servers and issue fake identities servers, cannot learn a single bit of each party's session key once the key exchange is successfully completed.

More precisely, we construct a ppt machine Ch^{KE}, called the challenger, that represents honest parties $(P_1, \ldots, P_n)$ and allows the attacker a fixed set of queries (see Figure 3). This challenger internally runs the 1W-AKE algorithms AKE. The definition basically states that an attacker breaks the 1W-AKE security if in the end it successfully distinguishes a randomly chosen session key from the actually established session key for an uncompromised session Ψ. For this challenge, the attacker sends a query $\mathsf{test}^P(\Psi)$ to Ch^{KE}. For triggering the initiation of a session, triggering the response to a key exchange, and for completing a key exchange, Ch^{KE} allows the attacker to send a query $\mathsf{send}^P(m)$. For compromising parties, the attacker can query three different types of messages. First, the attacker can ask party P to reveal the next public key that will be chosen with the query $\mathsf{reveal\_next}^P$. Second, the attacker can ask for a secret key for a corresponding public key X using the query $\mathsf{partner}^P(X)$. Third, the attacker can ask for the session key of a session Ψ with the query $\mathsf{sk\_reveal}^P(\Psi)$. Finally, the attacker can also register new long-term public keys pk_Q for unused identities Q with the query $\mathsf{establish\_certificate}(Q, pk_Q)$.

The challenger Ch^{KE} maintains several variables for every party P. A variable v for a party P is denoted as v^P. First, the challenger maintains the key exchange state $akest^P(\Psi)$ for a party P and a session Ψ. This key exchange state stores the ephemeral secret keys that will be erased after the key exchange is completed. Then, Ch^{KE} gets as input the public parameters *params*, typically containing the security parameter η and the name of the protocol. The challenger furthermore maintains for every party P the result state $resust^P(\Psi)$ of a completed session Ψ. This result state contains the established key, the peer party, which is $\star$ for the server P since the client is anonymous, and a state st that typically contains two vectors v_1, v_2 that contain the ephemeral public keys and the long-term keys used for establishing the session key of Ψ. In the case of ntor, v_1 contains the client's ephemeral key $X = v_1$ and v_2 contains the server's long-term key B and ephemeral key Y, i.e., $(Y, B) = v_2$. Recall that in the case of Ace v_1 contains the two ephemeral keys $(X_1, X_2) = v_1$ and v_2 is the same as in ntor, i.e., $(Y, B) = v_2$.

For characterizing those secret keys that are used in a key exchange and have not been leaked to the attacker yet, we introduce the notion of the attacker not being a *partner* to a ephemeral public key X. Formally, the attacker is a partner for a public value X if one of the following conditions hold true.

- X was not used yet.
- X is public key that the attacker registered using the query $\mathsf{establish\_certificate}(Q, X)$.
- X was the response of a query send^P or $\mathsf{reveal\_next}^P$ and there is a successive query $\mathsf{partner}^P(X)$.

We stress that unused values also include all values that are only chosen by the attacker; hence the attacker is with overwhelming probability a partner to all self-chosen values.

Moreover, we assume that if an attacker learns from a client one ephemeral key X_i of a session Ψ, then the attacker also learns the other ephemeral key X_{3-i} of that session. We ensure this by making a consistency check for all partnered values for every query. Even though this modification is particular to key exchange protocol in which the client sends two ephemeral keys, this modification looks natural to us.

Goldberg, Stebila, and Ustaoglu proposed a freshness notion for the challenge session, in order to prevent the attacker from trivially winning the game. We call their freshness condition *single value 1W-AKE freshness*. We say that a session Ψ at a party P is *single value 1W-AKE fresh* if the following two conditions hold:

1. Let $(k, Q, st) \leftarrow resust^P(\Psi)$ (see Figure 3). For every vector v_j in st there is at least one element X in v_j such that the attacker $\mathcal{M}$ is not a partner to X.
2. For the session Ψ such that $akest^P(\Psi) = (v, Q)$, the adversary did not issue $\mathsf{sk\_reveal}^Q(\Psi')$ for any Ψ' such that $akest^Q(\Psi') = (v, \star)$.

The protocol Ace presented in this work, uses two ephemeral keys for the client, i.e., $v_1 = (g^{x_1}, g^{x_2})$ and as ntor one ephemeral key g^y for the server and the long-term key g^b of the server, i.e., $v_2 = (g^y, g^b)$. The key is then deterministically derived from $g^{bx_1+yx_2}$. Since $g^{bx_1+yx_2}$ is computable for any attacker that is a partner to the pair (x_2, b) or the pair (x_1, y), we need to exclude these cases in order to prevent the attacker from trivially winning. We say that a session is *double value 1W-AKE fresh* if it is single value 1W-AKE fresh and the following condition holds:

3. Let $(k, Q, ((X_1, X_2), (Y, B))) \leftarrow resust^P(\Psi)$. The attacker is not a partner of the pair (X_1, Y) or (X_2, B).

We call a double value 1W-AKE fresh session a *fresh 1W-AKE* session.

As depicted in Figure 3, we consider two challengers $\mathsf{Ch}_0^{\text{KE}}$ and $\mathsf{Ch}_1^{\text{KE}}$. $\mathsf{Ch}_0^{\text{KE}}$ sends a randomly chosen key as a response, and $\mathsf{Ch}_1^{\text{KE}}$ sends the actually established key as a response. Upon successful key exchange with a server Q, a key k, and the transcript v_1, v_2, a client outputs a tuple $(k, Q, (v_1, v_2))$. A server outputs $(k, \star, (v_1, v_2))$ for denoting that the peer party is anonymous.

Definition 2 (1W-AKE-security). *Let η be the security parameter. A protocol π is said to be* 1W-AKE-secure

if, for all ppt adversaries $\mathcal{M}$, the following difference is negligible in η:

$$|\Pr[b^* \leftarrow \langle A(1^\eta), \mathsf{Ch}_0^{\text{KE}}(1^\eta)\rangle : b^* = 1] - \Pr[b^* \leftarrow \langle A(1^\eta), \mathsf{Ch}_1^{\text{KE}}(1^\eta)\rangle : b^* = 1]|$$

1W-Anonymity.

A one-way authenticated key exchange is able to provide anonymity for the unauthorized client; this client-side anonymity is called *1W-anonymity*. Formally, 1W-anonymity means that the attacker cannot link a key exchange through an anonymized channel (e.g., Tor) with a key exchange through a direct connection. More formally, we consider the following scenario.

The attacker can communicate with all parties directly, which is modeled by the 1W-AKE challenger $\mathsf{Ch}_1^{\text{KE}}$ (see Figure 3). In addition, the attacker chooses two candidate parties for a key exchange challenge session Ψ^* over an anonymous channel. This anonymous channel is modeled by the ppt machine Ch^{AN} (presented in Figure 4). Ch^{AN} selects one of the two candidate parties. Finally, the attacker has to guess which of the two parties has been selected in the challenge session.

In order to prevent the attacker from trivially learning the identity of the correct candidate, we have to exclude the cases in which the attacker peaks into the state of the candidate parties. Formally, we require that Ch^{AN} internally runs a copy of the 1W-AKE challenger Ch^{KE}. We denote the internal copy of Ch^{KE} as ICh^{KE} (see Figure 4).

Definition 3 (1W-anonymity). *Let η be the security parameter. Let M, N be ppt interactive turing machines. Let $v \leftarrow \langle A(1^\eta), M(1^\eta), N(1^\eta)\rangle$ denote the interaction between A and M and A and N and v be the output of A. A protocol* AKE *is said to be* 1W-anonymous *if, for all PPT adversaries $\mathcal{M}$, the following difference is negligible in η*

$$|\Pr[b^* \leftarrow \langle A(1^\eta), \mathsf{Ch}_0^{\text{AN}}(1^\eta), \mathsf{Ch}_1^{\text{KE}}(1^\eta)\rangle : b^* = 1] - \Pr[b^* \leftarrow \langle A(1^\eta), \mathsf{Ch}_1^{\text{AN}}(1^\eta), \mathsf{Ch}_1^{\text{KE}}(1^\eta)\rangle : b^* = 1]|$$

5.2 The Security of Ace

At this point, we are able to analyze the security of Ace. We first show that no information about the session key is leaked by proving 1W-AKE security for Ace. Then, we show that a Ace session cannot be linked to another Ace session by proving 1W-anonymity for Ace.

Lemma 1 (Ace is 1W-AKE secure). *If H_{st} is a collision resistant hash function,* Mac *is universally unforgeable against chosen message attacks (CMA-UF), and* H *is a random oracle, the protocol* Ace *is 1W-AKE-secure in the sense of Definition 2 under the GDH assumption.*[5]

More precisely, for every machine $\mathcal{M}$ that breaks the 1W-AKE security of Ace *with probability μ and runs in time t, there exists a bound $q \geq 1$ on the number of sessions and a machine S_q that breaks the GDH assumption with a probability of more than $\binom{2}{|P|}\mu/(q|P|)$ and runs in time $\mathsf{O}(t)$, where $|P|$ is the number of honest servers.*

[5]The GDH assumption states that the computational DH assumption holds even against an attacker that has access to a decisional DH oracle [14].

Proof. Let Game_1 be the original setup with the $\mathsf{Ch}_1^{\text{KE}}$ challenger against the attacker $\mathcal{M}$.

Game_2 is the faking game with the $\mathsf{Ch}_0^{\text{KE}}$ challenger: upon a $\mathsf{test}^P(\Psi)$-query $\mathsf{Ch}_0^{\text{KE}}$ sends a randomly chosen key k instead of the real key k_s for $(k_m, k_s) \leftarrow \mathsf{H}(g^{bx_1+yx_2}, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$. Since H is a random oracle k_m is completely independent of k_s; hence, no information about the challenge key k_s is leaked by using k_m for the Mac. Moreover, we show below that by the gap DH assumption $g^{bx_1+yx_2}$ cannot be computed from g^{x_1}, g^{x_2}, g^y, and g^b.

We construct a ppt reduction S_q against the GDH challenger for an attacker $\mathcal{M}$ that distinguishes Game_1 from Game_2 but only allows q session-queries. Moreover, this reduction S_q also simulates the random oracle. Let $P = \{P_1, \ldots, P_n\}$ be the set of parties. We show that there is a q such that S_q that solves the GDH problem with probability $\binom{2}{|P|}\mu/(q|P|)$ if $\mathcal{M}$ breaks the 1W-AKE security with probability μ, where $q \leq p(\eta)$ and p is the runtime polynomial of the attacker $\mathcal{M}$. Let (g, g^u, g^v) be the GDH challenge. Moreover, the runtime of S_q is asymptotically the same as the runtime of $\mathcal{M}$.

The reduction S_q answers all queries honestly, except for $\mathsf{partner}(g^u)$ or $\mathsf{partner}(g^v)$ queries. In these cases S_q aborts the simulation. If the attacker stops, S_q draws a random group element and sends it as a blind guess to the GDH challenger. The simulator S_q cannot compute $g^{bx_1+yx_2}$ if the GDH challenge exponent v equals b. Since, however, $g^{bx_1+yx_2}$ is never sent in plain but always hashed and S_q also simulates the random oracle, these hashes can be faked without knowing the input.

Besides, the simulator does the following:

S_q: upon initialization
ask the GDH challenger for a DH tuple (g, g^u, g^v)
draw $b \leftarrow_R \{0, 1\}$
if $b = 1$ **then**
  draw $i \leftarrow_R \{1, \ldots, q\}$
else
  draw $i \leftarrow_R \{1, \ldots, |P|\}$
  replace $pk_{P_i} = g^b$ of party P_i with g^v
  draw $j \leftarrow_R \{1, \ldots q\}$

S_q: upon $\mathsf{send}^P(params, Q)$:
/* if the client is called for the first protocol message */
if $b = 1$ and it is the ith session **then**
  replace g^{x_2} with g^u
  honestly choose g^{x_1}
if $b = 0 \wedge Q = P_i$ and it is the jth session **then**
  replace g^{x_1} with g^u
  honestly choose g^{x_2}
proceed as in $(m, st, \Psi) \leftarrow \mathsf{Init}(Q, (\eta, \mathsf{new\_session}, \mathsf{Ace}), cs)$
/* recall that cs is maintained by the challenger */
$akest^P(\Psi) \leftarrow (Q, st)$; send (m, Ψ)

S_q: upon $\mathsf{send}^P(\Psi, (\eta, \mathsf{Ace}, g^{x_1}, g^{x_2}))$ and $akest^P(\Psi) = \perp$
/* if the server is called */
if $b = 1$ and Ψ is the ith session **then**
  replace g^y with g^v
  honestly choose g^b
  draw $r = (k_m, k_s)$ at random from the range of RO
  store $faked(g^{x_1}, g^{x_2}, g^v, g^b, \mathsf{Ace}) \leftarrow r$
else
  replace $pk_{P_i} = g^b$ of party P_i with g^v
  honestly choose g^y

upon $\mathsf{start}(i, j, params, Q)$**:** *(one time query)*
  if $i \neq j$ **then**
    if $b = 1$ **then** $i^* \leftarrow i$ **else** $i^* \leftarrow j$
    send $\mathsf{send}^{P_{i^*}}(params, Q)$ to $\mathsf{ICh}_1^{\mathrm{KE}}(1^\eta)$
    wait for the response (Ψ^*, m'); send m' to $\mathcal{M}$

upon $\mathsf{send}(m)$**:**
  forward $\mathsf{send}^{P_{i^*}}$ to $\mathsf{ICh}_1^{\mathrm{KE}}(1^\eta)$

upon reveal_next**:**
  forward $\mathsf{reveal\_next}^{P_{i^*}}$ to $\mathsf{ICh}_1^{\mathrm{KE}}(1^\eta)$

upon sk_reveal**:**
  forward $\mathsf{sk\_reveal}^{P_{i^*}}(\Psi^*)$ to $\mathsf{ICh}_1^{\mathrm{KE}}(1^\eta)$

upon $\mathsf{partner}(X)$**:**
  forward $\mathsf{partner}^{P_{i^*}}(X)$ to $\mathsf{ICh}_1^{\mathrm{KE}}(1^\eta)$

Figure 4: The anonymizing machine $\mathsf{Ch}_b^{\mathrm{AN}}(1^\eta)$: $\mathsf{ICh}_1^{\mathrm{KE}}(1^\eta)$ is an internally emulated copy of $\mathsf{Ch}_1^{\mathrm{KE}}(1^\eta)$

  draw $r = (k_m, k_s)$ at random from the range of RO
  store $\mathit{faked}(g^{x_1}, g^{x_2}, g^y, g^v, \mathsf{Ace}) \leftarrow r$
proceed as in $(m', (k, \star, st), \Psi) \leftarrow \mathsf{Resp}(sk_P, P, m, cs)$
$\mathit{resust}^P(\Psi) \leftarrow (k, \star, st)$; send m'

S_q **upon** $\mathsf{send}^P(\Psi, (\eta, \mathsf{Ace}, Y, t_Q))$ **and** $\mathit{akest}^P(\Psi) \neq \bot$
/* if the client is called with the response of the server */
lookup $(Q, (x_1, x_2), (g^{x_1}, g^{x_2})) \leftarrow \mathit{akest}^P(\Psi)$
check for a valid pk_Q
if $\mathit{faked}(g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$ is defined **then**
  lookup $(k_m, k_s) = r \leftarrow \mathit{faked}(g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$
  if $g^v = g^b$ $(pk_Q = g^b)$ **then**
    query $(g, g^b, g^{x_1}, Z/g^{yx_2})$ to the DDH oracle
  else if $g^v = Y = g^y$ **then**
    query $(g, g^y, g^{x_2}, Z/g^{bx_1})$ to the DDH oracle
  if the DDH oracle confirms **then**
    program $\mathrm{RO}(Z, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace}) := r$
proceed as in $(k, Q, st) \leftarrow \mathsf{CompKey}(pk_Q, m, \Psi, (Q, st))$
erase $\mathit{akest}^P(\Psi)$; $\mathit{resust}^P(\Psi) \leftarrow (k, Q, st)$

S_q **simulating the RO: upon** $(Z, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$
if $(g^u, g^v) = (g^{x_1}, g^b)$ **then**
  query $(g, g^u, g^v, Z/g^{yx_2})$ to the DDH oracle
  if the DDH oracle confirms **then**
    send Z/g^{yx_2} as a guess and stop
else if $(g^u, g^v) = (g^{x_2}, g^y)$ **then**
  query $(g, g^u, g^v, Z/g^{bx_1})$ to the DDH oracle
  if the DDH oracle confirms **then**
    send Z/g^{bx_1} as a guess and stop
if $\mathrm{RO}(Z, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace}) = r$ is defined **then**
  respond with r
else
  draw $r = (k_m, k_s)$ at random from the range of RO
  program $\mathrm{RO}(Z, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace}) := r$

Recall that we required the challenge session to be fresh. Let (g^{x_1}, g^{x_2}) be the ephemeral keys of the client, g^y be the ephemeral key of the server, and g^b be the long-term key of the server. By the freshness of the challenge session, we conclude that with overwhelming probability the attacker at most a partner to (x_2, y) or to (x_1, b). Hence, it suffices to consider these two cases in which the attacker is not a partner to (x_2, y) or not to (x_1, b).

If $\mathcal{M}$ is not a partner to (x_1, b), then S_q either knows x_2 or y and can hence compute g^{yx_2}. Moreover, with probability $1/(q|P|)$ we have $b = v$ and $x_1 = u$. Then, the simulator guesses g^{uv} correctly if Z is the shared secret, i.e., if $Z = g^{bx_1+yx_2}$, since

$$Z/g^{yx_2} = g^{bx_1+yx_2-yx_2} = g^{uv+yx_2-yx_2} = g^{uv}.$$

If $\mathcal{M}$ is not a partner to (x_2, y), then S_q knows x_1 or b and can hence compute g^{bx_1}. We stress that the attacker cannot send a maliciously chosen y' (such as $y' = 0$) because a successful forgery of the MAC tag would lead to an attack against the CMA-UF property of Mac. Then, with probability $1/q$ we have $x_2 = u$ and $y = v$. Then, again the simulator guesses g^{uv} correctly if Z is the shared secret, i.e., if $Z = g^{bx_1+yx_2}$, since

$$Z/g^{bx_1} = g^{bx_1+yx_2-bx_1} = g^{bx_1+uv-bx_1} = g^{uv}.$$

Note that for $z = uv$, S_q is indistinguishable from Game_1 as long as $\mathsf{partner}(g^u)$ and $\mathsf{partner}(g^v)$ is not queried. Similarly for a randomly chosen z, S_q is indistinguishable from Game_2 as long as $\mathsf{partner}(g^u)$ and $\mathsf{partner}(g^v)$ is not queried. Below, we denote this event that $\mathsf{partner}(g^u)$ and $\mathsf{partner}(g^v)$ is not queried as T. The probability that T occurs is more than $\binom{2}{|P|} = \frac{2}{|P|(|P|-1)}$, where P is the set of parties.

We conclude that if the attacker can distinguish Game_1 from Game_2 with more than negligible probability, then the attacker queried the random oracle with

$$(g^{bx_1+yx_2}, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace}).$$

The overall winning probability of the simulator S_q can, hence, be computed as follows. Let E_1 be the event that the attacker is not a partner to (x_2, y) and E_2 the event that the attacker is not a partner to (x_1, b). Recall T is the event that $\mathsf{partner}(g^u)$ and $\mathsf{partner}(g^v)$ has not been queried. Moreover, let W be the event that the simulator S_q wins against the GDH challenger, and μ be the probability that the attacker distinguishes Game_1 from Game_2. Then, we get

$$\begin{aligned}
\Pr[W] &= \Pr[E_1] \cdot \Pr[W \mid E_1] + \Pr[E_2] \cdot \Pr[W \mid E_2] \\
&= \Pr[E_1] \cdot \frac{\mu}{q|P|} \cdot \Pr[T] + \Pr[E_2] \cdot \frac{\mu}{q} \cdot \Pr[T] \\
&\geq \Pr[E_1] \cdot \frac{\mu}{q|P|} \cdot \binom{2}{|P|} + \Pr[E_2] \cdot \frac{\mu}{q} \cdot \binom{2}{|P|} \\
&\overset{(1)}{\geq} \binom{2}{|P|} \frac{\mu}{(q|P|)} = \frac{2\mu}{q|P|^2(|P|-1)}
\end{aligned}$$

where (1) holds since $\Pr[E_1] + \Pr[E_2] = 1$.

Hence, S_q breaks the GDH game with probability more than $\binom{2}{|P|}\mu/(q|P|)$ if the attacker distinguishes Game_1 from

Game_2 with probability μ. In particular, if the GDH assumption holds Game_2 is indistinguishable from the real setting Game_1, and the 1W-AKE security holds. □

The proof for the 1W-anonymity of Ace is almost exactly the same as the proof of the 1W-anonymity of ntor [7]. Therefore, we refer for the proof to their work and only state the result.

LEMMA 2 (Ace IS 1W-ANONYMOUS). *The* Ace *protocol is 1W-anonymous in the sense of Definition 3.*

6. CONCLUSION AND FUTURE WORK

The Tor project is currently considering to revise the key-exchange protocol used for establishing the circuits. Ace is a novel and provably secure 1W-AKE protocol, and we propose it for use in the Tor's circuit establishment protocol. Compared to the current candidate for Tor's new key-exchange protocol, Ace offers a client-side efficiency improvement of 46% and a server-side efficiency improvement of nearly 19%. Even though Ace requires the client to send one additional group element, it does not produce any communication overhead in Tor as Ace only occupies some of the unused space in a Tor packet, in the ECC setting. Given that the ECC setting is under consideration for the Tor system, the improved computational efficiency, and the proven security properties make our 1W-AKE an ideal candidate for use in the Tor protocol.

Acknowledgments

We thank Roger Dingledine and Nick Mathewson for motivating preliminary discussions, and Berkant Ustaoglu, Robert Ransom, and the anonymous reviewers for their valuable comments on an earlier draft of the paper.

7. REFERENCES

[1] R. M. Avanzi. The Complexity of Certain Multi-Exponentiation Techniques in Cryptography. *J. Cryptology*, 18(4):357–373, 2005.

[2] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi. Provably secure and practical onion routing. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF)*, 2012.

[3] D. J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Proc. 9th Conference on Theory and Practice of Public-Key Cryptography (PKC)*, pages 207–228, 2006.

[4] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-Speed High-Security Signatures. In *CHES' 11*, pages 124–142, 2011. `http://ed25519.cr.yp.to/`.

[5] D. Catalano, D Fiore, and R. Gennaro. Certificateless onion routing. In *Proc. 16th ACM Conference on Computer and Communication Security (CCS)*, pages 151–160, 2009.

[6] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. 13th USENIX Security Symposium (USENIX)*, pages 303–320, 2004.

[7] I. Goldberg, D. Stebila, and B. Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, pages 1–25, 2012. Proposal for Tor: `https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/ideas/xxx-ntor-handshake.txt`.

[8] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing. In *Proc. 7th Privacy Enhancing Technologies Symposium (PETS)*, pages 95–112, 2007.

[9] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing with Improved Forward Secrecy. *ACM Trans. Inf. Syst. Secur.*, 13(4):29, 2010.

[10] N. Mathewson. Another key exchange algorithm for extending circuits: alternative to ntor? The tor-dev mailing list, 2012. `https://lists.torproject.org/pipermail/tor-dev/2012-August/003901.html`.

[11] N. Mathewson. Rump Session Talk on Tor. 12th Privacy Enhancing Technologies Symposium (PETS), 2012.

[12] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 1997.

[13] D. M'Raïhi and D. Naccache. Batch Exponentiation: A Fast DLP-Based Signature Generation Strategy. In *ACM Conference on Computer and Communications Security (CCS '96)*, pages 58–61, 1996.

[14] T. Okamoto and D. Pointcheval. The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In *PKC' 01*, pages 104–118, 2001.

[15] L. Øverlier and P. Syverson. Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services. In *Proc. 7th Privacy Enhancing Technologies Symposium (PETS)*, pages 134–152, 2007.

[16] V. Shoup. On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, Report 1999/012, 1999. Available as Cryptology ePrint Archive, Report 1999/012 `http://eprint.iacr.org/1999/012`.

[17] J. Solinas. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, 2001. Available at `http://cacr.uwaterloo.ca/techreports/2001/corr2001-41.ps`.

[18] The Tor Project. `https://www.torproject.org/`, 2003. Accessed Nov 2011.

APPENDIX

In Figure 5, we illustrate the important protocols that we discussed in the paper.

The Tor Authentication Protocol (TAP)

(no long-term key)	Client		Server	(long-term keys (sk_B, pk_B))
	$x \leftarrow_R G$	$\xrightarrow{\mathsf{Enc}(pk_B, g^x)}$	$y \leftarrow_R G$	
		$\xleftarrow{g^y}$		
	$(g^y)^x = g^{yx}$		$g^{xy} = (g^x)^y$	(established shared secret g^{xy})

The A-DHKE Protocol

(no long-term key)	Client		Server	(long-term keys (sk_B, pk_B))
	$x \leftarrow_R G$	$\xrightarrow{g^x}$	$y \leftarrow_R G$	
		$\xleftarrow{g^y, \mathsf{Sig}(sk_B, g^x, g^y)}$		
	$(g^y)^x = g^{yx}$		$g^{xy} = (g^x)^y$	(established shared secret g^{xy})

The ØS Protocol

(no long-term key)	Client		Server	(long-term keys (b, g^b))
	$x \leftarrow_R G$	$\xrightarrow{g^x}$	$y \leftarrow_R G$	
		$\xleftarrow{g^y}$		
	$(g^y g^b)^x = g^{(b+y)x}$		$g^{x(b+y)} = (g^x)^{y+b}$	(established shared secret $g^{x(b+y)}$)

An Attack on the ØS Protocol

(no long-term key)	Client		$\mathcal{M}$	(long-term keys (b, g^b))
	$x \leftarrow_R G$	$\xrightarrow{g^x}$	$y \leftarrow_R G$	
		$\xleftarrow{g^y / g^b = g^{y-b}}$		
	$(g^{y-b} g^b)^x = g^{yx}$		$g^{xy} = (g^x)^y$	(established shared secret g^{xy})

The ntor Protocol

(no long-term key)	Client		Server	(long-term keys (b, g^b))
	$x \leftarrow_R G$	$\xrightarrow{g^x}$	$y \leftarrow_R G$	
		$\xleftarrow{g^y}$		
	$\mathsf{H}((g^y)^x, (g^b)^x)$ $= \mathsf{H}(g^{yx}, g^{bx})$		$\mathsf{H}((g^x)^y, (g^x)^b) =$ $\mathsf{H}(g^{xy}, g^{xb})$	(established session key $\mathsf{H}(g^{xy}, g^{xb})$)

The Ace Protocol

(no long-term key)	Client		Server	(long-term keys (b, g^b))
	$x_1, x_2 \leftarrow_R G$	$\xrightarrow{g^{x_1}, g^{x_2}}$	$y \leftarrow_R G$	
		$\xleftarrow{g^y}$		
	$(g^b)^{x_1}(g^y)^{x_2}$ $= g^{x_1 b + x_2 y}$		$(g^{x_1})^b (g^{x_2})^y =$ $g^{x_1 b + x_2 y}$	(established shared secret $g^{x_1 b + x_2 y}$)

Figure 5: A comparative overview over all discussed protocols: G is the exponent group. For the sake of readability, we neglected the session information used for the key derivation and the key confirmation message.