Attribute-based Key Exchange

Alex J. Malozemoff
Galois

Joint with: Vladimir Kolesnikov, Hugo Krawczyk, Yehuda Lindell, Tal Rabin

Published at CCS 2016

TPMPC 2017
April 7, 2017
Consider the following scenario...

**Goal:** Establish secure channel with agent in green zone

**Agent goal:** Establish secure channel, keep location hidden
Consider the following scenario...

χ: \( x_U = 48.2, \ y_U = 16.4 \)

χ': \( x'_U = 48.1, \ y'_U = 17.1 \)

\[ P(x_U, y_U) := (x_U - x_S)^2 + (y_U - y_S)^2 < d^2 \]
Consider the following scenario...

\[ P(x_U, y_U) := (x_U - x_S)^2 + (y_U - y_S)^2 < d^2 \]
Consider the following scenario...

\[ P(x_U, y_U) := (x_U - x_S)^2 + (y_U - y_S)^2 < d^2 \]

Viewed as bitstrings: \( \chi \in \{0, 1\}^m \)

Attributes

Private

Public

Policy

Viewed as (Boolean) circuit
Contributions

1. Define
   • Attribute-based key exchange (ABKE)
   • **Goal**: secure channel *if and only if* policy satisfied
   • **Also want**: attribute privacy, unlinkability, collusion resistance
   • UC security definition

2. Design
   • Build using garbled circuits, *attribute-selective encryption*
   • Supports *arbitrary* policies

3. Implement
   • 241 ms server computation time
   • 81 ms client computation time  
   } for policy with **100,000** gates, **200** attributes
ABKE: possible solutions

1. Attribute-based credentials
   • **Goal:** identification, access control
   • Constructions based on zero-knowledge proofs
   • Cannot handle complicated policies

2. Attribute-based encryption
   • **Goal:** only clients that satisfy policy can decrypt
   • Cannot handle complicated policies (efficiently)

IBM Idemix
ABKE: similar notions

1. Credential authenticated key exchange [CamenischCasatiGroßShoup10]
   • Adaptation of attribute-based credentials to key exchange setting

2. Attribute-based authenticated key exchange [GorantlaBoydNieto10]
   • Adaptation of attribute-based encryption to key exchange setting

Our approach:
• New (more flexible) definition
• Allows for efficient construction based on garbled circuits
Attribute-based key exchange (ABKE)

CA: distributes attributes to clients

: holds policy

: holds attributes

Derive shared key ⇔ attributes satisfy policy

Features:
- Attribute privacy
- Unlinkability
- Collusion resistance

cannot learn ’s attributes

cannot correlated two interactions with

and cannot combine attributes
ABKE definition

Captures attribute privacy, unlinkability, collusion resistance

Requires anonymous channels
Building ABKE

Based on **zero-knowledge using garbled circuits** protocol [JawurekKerschbaumOrlandi13]

Combine with new primitive: **attribute selective encryption**

Two building blocks:
1. (Verifiable) garbled circuits
2. **Oblivious transfer**
Building block: garbled circuit (GC)

Way to construct “garbled” version of $F$

$F \Rightarrow$

Evaluator learns nothing about internal wires

Values on circuit wires mapped to “wire labels”

$\hat{w}_0, \hat{w}_1$

\forall gates: encrypt output wire label using input wire labels
Building block: *verifiable* garbled circuit (GC)

\[ \mathcal{G} = (\text{Gb}, \text{Ev}, \text{Ve}) \]

\[ \text{Gb}(C) \rightarrow \text{GC}, \{\hat{w}_0, \hat{w}_1\} \]

\[ \text{Ev}(\text{GC}, \{\hat{w}_{x_i}\}) \rightarrow \{\hat{w}_{z_j}\} \quad C(x) = z \]

\[ \text{Ve}(C, \text{GC}, \{\hat{w}_0, \hat{w}_1\}) \rightarrow \text{accept/reject} \quad \text{is GC a garbling of} \ C \]
Building block: oblivious transfer (OT)

Sender: Cannot learn anything about $b$
Receiver: Cannot learn anything about $m_{1-b}$
Zero-knowledge using garbled circuits \[ \text{[JawurekKerschbaumOrlandi13]} \]

\[ F, x, y \]

\[ \{\hat{w}_x\} \]

Prover

\[ \text{GC, } \{\hat{w}_0, \hat{w}_1\} \leftarrow \text{Gb}(F(\cdot) \equiv y) \]

GC

\[ F, y \]

Verifier

\[ \text{oblivious transfer} \]

\[ \{\hat{w}_0, \hat{w}_1\} \]

\[ \text{Ev(GC, } \{\hat{w}_x\} \rightarrow \hat{z} \]

\[ \text{Commit} (\hat{z}) \]

\[ \{\hat{w}_0, \hat{w}_1\} \]

\[ \text{Ve}(F, \text{GC, } \{\hat{w}_0, \hat{w}_1\} \equiv \text{accept} \]

\[ \text{Decommit}(\hat{z}) \]

Output \[ \begin{cases} 1 & \text{if } z = 1 \\ 0 & \text{otherwise} \end{cases} \]
Attribute-based key exchange

\[ P, \chi \]

\( \{ \hat{w}_{\chi_i} \} \)

\[ \text{Ev}(GC, \{ \hat{w}_{\chi_i} \}) \rightarrow \hat{z} \]

\[ \text{Commit}(\hat{z}) \]

\[ \{ \hat{w}_0, \hat{w}_1 \} \]

\[ \text{Decommit}(\hat{z}) \]

\[ \text{Coin tossing} \]

\[ \text{accept} \]

\[ \text{abort if } z = 0 \]

\[ \text{Client} \]

\[ \text{GC} \]

\[ \{ \hat{w}_0, \hat{w}_1 \} \leftarrow \text{Gb}(P(\cdot)) \]

\[ \text{Server} \]
Attribute selective encryption (ASE)

Similar to *attribute-based encryption*:
- Decrypt based on attribute vector

Similar to *oblivious transfer*:
- Receive based on attribute vector bits

\[
(x_1, 0 \ldots x_m, 0) \quad \xrightarrow{ASE} \quad \chi
\]

\[
(x_{1,1} \ldots x_{m,1})
\]

\[
\text{ASE.Setup}(m) \rightarrow mvk, msk
\]

\[
\text{ASE.Enc}(pk'_\chi, \bullet) \rightarrow c
\]

\[
\text{ASE.GenCert}(msk, \chi) \rightarrow pk_\chi, sk_\chi
\]

\[
\text{ASE.Verify}(mvk, pk_\chi) \rightarrow \{0, 1\}
\]

\[
\text{ASE.Dec}(sk'_\chi, c) \rightarrow
\]

\[
\text{ASE.Unlink}(pk_\chi, sk_\chi) \rightarrow pk'_\chi, sk'_\chi
\]
Attribute selective encryption (ASE)

Security properties:
• Attribute privacy
• Collusion resistance
• Unlinkability

$p_k \chi$ hides $\chi$
Colluding clients cannot produce valid $p_k \chi$
Server cannot link two interactions with same $p_k \chi$

Provide three game-based definitions:
1. Attribute privacy
2. Unlinkability
3. Collusion resistance

"Standard" tailored to ABKE use case
ASE security: attribute privacy

\[ C \]

\[ mvk, msk \leftarrow \text{Setup}(m) \]

\[ mvk \]

\[ b \in_R \{0, 1\} \]

\[ \chi \]

\[ \begin{cases} 
\text{if } b = 0 & pk, sk \leftarrow \text{GenCert}(msk, \chi) \\
\text{if } b = 1 & pk, sk \leftarrow \text{GenCert}^*(msk) 
\end{cases} \]

\[ pk \]

\[ \text{output } b' \]

\[ A \]

\[ A \text{ wins if } b' = b \]
ASE security: unlinkability

\[ C \]

\[ mvk, msk \leftarrow \text{Setup}(m) \]

\[ mvk \]

\[ \chi \]

\[ pk_0, sk_0 \leftarrow \text{GenCert}(msk, \chi) \]

\[ pk_1, sk_1 \leftarrow \text{Unlink}(pk_0, sk_0) \]

\[ b \in_R \{0, 1\} \]

\[ pk_b \]

\[ \text{output } b' \]

\[ A \text{ wins if } b' = b \]
ASE security: collusion resistance

\[ C \]  \[ A \]

\[ mvk, msk \leftarrow \text{Setup}(m) \]

\[ \begin{array}{c}
\text{mvk} \\
\hline
\text{pk}
\end{array} \]

\[ c \leftarrow \text{Enc}(pk, \vec{x}) \]

\[ c \]

\[ \text{output set } \mathcal{M} \]

\[ \mathcal{A} \text{ wins if and only if:} \]

- \[ \text{Verify}(pk) = 1 \]

- \[ \exists \mathcal{M}' \subseteq \mathcal{M} \text{ s.t. either } \]
  \[ \begin{cases} 
  (1) & m \in \mathcal{M}' \implies m \in \{x_i, x[i]\}, x \not\in \mathcal{X} \\
  (2) & s, s' \in \mathcal{M}' \text{ s.t. } s = x_{i,0}, s' = x_{i,1}
  \end{cases} \]
Attribute-based key exchange

\[ P, \chi \]

\[ \{ \hat{w}_{\chi_i} \} \]

\[ \text{Ev}(\text{GC}, \{ \hat{w}_{\chi_i} \}) \rightarrow \hat{z} \]

\[ \text{Commit}(\hat{z}) \]

\[ \{ \hat{w}_0, \hat{w}_1 \} \]

\[ \text{Ve}(P, \text{GC}, \{ \hat{w}_0, \hat{w}_1 \}) \]

\[ ? \text{ accept} \]

\[ \text{Decommit}(\hat{z}) \]

\[ \text{Coin tossing} \]

\[ \text{GC, } \{ \hat{w}_0, \hat{w}_1 \} \leftarrow \text{Gb}(P(\cdot)) \]

\[ \text{abort if } z = 0 \]
Attribute-based key exchange

\[ P, \chi, pk_{\chi}, sk_{\chi} \]

\[ \text{GC}, \{\hat{w}_0, \hat{w}_1\} \leftarrow \text{Gb}(P(\cdot)) \]

\[ (pk'_{\chi}, sk'_{\chi}) \leftarrow \text{ASE.Unlink}(pk_{\chi}, sk_{\chi}) \]

\[ pk'_{\chi} \]

\[ \text{ASE.Verify}(pk'_{\chi}) \]

\[ c \leftarrow \text{ASE.Enc}(pk'_{\chi}, \{\hat{w}_0, \hat{w}_1\}) \]

\[ \{\hat{w}_{\chi_i}\} \leftarrow \text{ASE.Dec}(sk'_{\chi}, c) \]

\[ \text{Ev}(\text{GC}, \{\hat{w}_{\chi_i}\}) \rightarrow \hat{z} \]

\[ \text{Commit}(\hat{z}) \]

\[ : \]
Constructing ASE

Two constructions:
1. Based on extractable linearly homomorphic (ELH) signatures
2. Based on identity-based encryption (+ ELH signatures)

Tools/assumptions needed:
1. Bilinear maps
2. Knowledge-of-exponent
3. Random oracle (in ABKE)
Constructing ASE: efficiency

Two constructions:
1. Based on *extractable linearly homomorphic (ELH) signatures*
2. Based on *identity-based encryption (+ ELH signatures)*

- **Unlink:** $6 + 2m$ exponentiations
- **Verify:** $6 + 2m$ pairings
- **Encrypt:** $4m$ exponentiations
- **Decrypt:** $m$ exponentiations

- **Unlink:** $2$ exponentiations
- **Verify:** $2$ pairings
- **Encrypt:** $2m$ pairings, $2m$ exponentiations
- **Decrypt:** $m$ pairings, $m$ exponentiations

$m$: # attributes
Extractable linearly homomorphic (ELH) signatures

Signature scheme with **two** properties:
1. Signatures are *linearly homomorphic*: 
   \[ \text{Sig}(m_1 m_2) = \text{Sig}(m_1) \text{Sig}(m_2) \]
2. Exists “extractor” which given \( \{ g_i, g_i^{x_i} \} \) “knows” \( \{ x_i \} \)

Example ELH signature scheme: Variant of Boneh-Lynn-Shacham [BLS04]

**Note:** 
\[ \text{Sig}(m_1 m_2) = (m_1 m_2)^x = m_1^x m_2^x = \text{Sig}(m_1) \text{Sig}(m_2) \]

Public key: \( h, h^x \)
Secret key: \( x \)

Sign(\( m \)) := \( m^x = \sigma \)

Verify(\( m, \sigma \)) := \( e(\sigma, h) \stackrel{?}{=} e(m, h^x) \)
ASE from ELH signatures

For simplicity, assume $m = 1$

$\text{ASE.Setup}(m) :=$

$$mvk = \begin{cases} v_{k_g} \\ v_{k_h} \\ v_{k_u} \end{cases}$$

$$msk = \begin{cases} s_{k_g} \\ s_{k_h} \\ s_{k_u} \\ s_k \end{cases}$$

Verification/secret key for ELH signature scheme
**ASE from ELH signatures**

Assume $\chi \in \{0, 1\}$

$$\text{ASE.GenCert}(msk, \chi) := pk_\chi \quad sk_\chi$$

$m_{sk}$ =

\[
\begin{cases} 
    s_{kg} \\
    s_{kh} \\
    s_{ku} \\
    sk \\
\end{cases}
\]

- $g, h, u$
- $r \in \mathbb{Z}_q^*$
- $e = \frac{g^r}{h^r}$ if $\chi = 0$
- otherwise

- $\text{Sign}_{sk_g}(g)$, $\text{Sign}_{sk_h}(h)$, $\text{Sign}_{sk_u}(u)$
- $\text{Sign}_{sk}(ue)$
ASE from ELH signatures

\[ g, h, u \quad r \in \mathbb{Z}_q^* \quad e = \frac{g^r}{h^r} \quad \text{if } \chi = 0 \quad \text{otherwise} \]

\[ \text{Sign}_{sk_g}(g), \text{Sign}_{sk_h}(h), \text{Sign}_{sk_u}(u), \text{Sign}_{sk}(ue) \]

\[ \text{ASE.Verify}(mvk, pk_\chi) := \text{Check signatures are valid} \]

\[ mvk = \left\{ vk_g, vk_h, vk_u, vk \right\} \]
ASE from ELH signatures

$$g, h, u \quad r \in \mathbb{Z}_q^* \quad e = \frac{g^r}{h^r} \quad \text{if } \chi = 0$$

$$\text{if } \chi = 0 \quad \text{otherwise}$$

$$\text{Sign}_{sk_g}(g), \text{Sign}_{sk_h}(h), \text{Sign}_{sk_u}(u) \quad \text{Sign}_{sk}(ue)$$

ASE.Unlink$$ (pk_{\chi}, sk_{\chi}) :=

$$g^{r'}, h^{r'}, u^{r'}, e^{r'} \quad r' \cdot r$$

$$\text{(Sign}_{sk_g}(g))^{r'}, \text{(Sign}_{sk_h}(h))^{r'}, \text{(Sign}_{sk_u}(u))^{r'} \quad \text{(Sign}_{sk}(ue))^{r'}$$
ASE from ELH signatures

\[ g, h, u \quad r \in \mathbb{Z}_q^* \quad e = \begin{cases} g^r & \text{if } \chi = 0 \\ h^r & \text{otherwise} \end{cases} \]

\[ \text{Sign}_{sk_g}(g), \text{Sign}_{sk_h}(h), \text{Sign}_{sk_u}(u) \quad \text{Sign}_{sk}(ue) \]

\[
\text{ASE.Enc}(pk_x, \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}) := \begin{cases} s, t \in_R \mathbb{Z}_q^* \\ (g^s, x_0 \cdot e^s) \\ (h^t, x_1 \cdot e^t) \end{cases}
\]
ASE from ELH signatures

\[ g, h, u \quad r \in \mathbb{Z}_q^* \quad e = \frac{g^r}{h^r} \quad \text{if } \chi = 0 \]

\[ \text{otherwise} \]

\[ \text{Sign}_{sk_g}(g), \text{Sign}_{sk_h}(h), \text{Sign}_{sk_u}(u), \text{Sign}_{sk}(ue) \]

\[ \text{ASE.Dec}(sk_{\chi}, (c_0, c_1)) := \]

\[ \begin{cases} 
  c_{0,1}/c_{0,0}^r & \text{if } \chi = 0 \\
  c_{1,1}/c_{1,0}^r & \text{if } \chi = 1 
\end{cases} \]

\[ = \begin{cases} 
  x_0 \cdot e^s/g^{sr} & \text{if } \chi = 0 \\
  x_1 \cdot e^t/h^{tr} & \text{if } \chi = 1 
\end{cases} \]
Implementation

ABKE using ASE from ELH signatures

URL: https://www.github.com/amaloz/abke

- Random oracle: instantiated using SHA-1 \(^{(\text{oops})}\)
- Garbling library: libgarble \((https://github.com/amaloz/libgarble)\)
- Pairing library: RELIC, BN-P256 curve

C garbling library based on JustGarble

1.88 ms per pairing
160 \(\mu\)s per exponentiation
Experiments

Two experiments:
1. (Unoptimized) ABKE execution
2. Optimized ABKE execution

Setup:
• Intel Core i5-4210H CPU, 2.90 GHz (1 processor)
• Ran over localhost, emulated WAN
  • 33 ms latency, 200 Mbps bandwidth

URL: https://www.github.com/amaloz/abke
Performance: computation

- 944 ms
- 159 ms

Intel Core i5-4210H CPU, 2.9 GHz
Performance: *optimized* computation

1. Moving computation offline
2. Batch verification

Intel Core i5-4210H CPU, 2.9 GHz
Performance: batch verification

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLS* sign</td>
<td>522,767</td>
</tr>
<tr>
<td>BLS* verify</td>
<td>12,316,919</td>
</tr>
<tr>
<td>BLS* batch verify</td>
<td>22,635,625</td>
</tr>
</tbody>
</table>

Batch verification of 10 messages $\implies$ 5.4x improvement

**BLS**: ELH signature scheme
Performance: computation breakdown

<table>
<thead>
<tr>
<th>Step</th>
<th>S</th>
<th>S [opt]</th>
<th>P</th>
<th>P [opt]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (Gb)</td>
<td>5 ms</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3 (Unlink)</td>
<td>—</td>
<td>—</td>
<td>78 ms</td>
<td>—</td>
</tr>
<tr>
<td>4 (Vrfy)</td>
<td>857 ms</td>
<td>159 ms</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4 (Enc)</td>
<td>82 ms</td>
<td>82 ms</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>5 (Dec)</td>
<td>—</td>
<td>—</td>
<td>28 ms</td>
<td>28 ms</td>
</tr>
<tr>
<td>5 (Ev)</td>
<td>—</td>
<td>—</td>
<td>3 ms</td>
<td>3 ms</td>
</tr>
<tr>
<td>6 (commit)</td>
<td>—</td>
<td>—</td>
<td>&lt;1 ms</td>
<td>&lt;1 ms</td>
</tr>
<tr>
<td>7 (Enc)</td>
<td>—</td>
<td>—</td>
<td>42 ms</td>
<td>42 ms</td>
</tr>
<tr>
<td>7 (Ve)</td>
<td>—</td>
<td>—</td>
<td>8 ms</td>
<td>8 ms</td>
</tr>
<tr>
<td>8 (cointoss)</td>
<td>&lt;1 ms</td>
<td>&lt;1 ms</td>
<td>&lt;1 ms</td>
<td>&lt;1 ms</td>
</tr>
<tr>
<td>Total</td>
<td>944 ms</td>
<td>241 ms</td>
<td>159 ms</td>
<td>81 ms</td>
</tr>
</tbody>
</table>

- **#AND gates: 100,000**
- **#attributes: 200**

Intel Core i5-4210H CPU, 2.9 GHz
Performance: communication

33 ms latency, 200 Mbps bandwidth
Recall the following scenario...

\[ \chi: x_U = 48.2, \ y_U = 16.4 \]

20,000 gates (assuming 32-bit inputs)

\[ x_S, y_S, d \]

\[ P(x_U, y_U) := (x_U - x_S)^2 + (y_U - y_S)^2 < d^2 \]

\[ \chi': x'_U = 48.1, \ y'_U = 17.1 \]
Future work

1. ASE definition “tied” to ABKE construction
   • Q: Can this be decoupled?

2. Current ASE constructions rely on hodgepodge of assumptions
   • Bilinear maps, knowledge-of-exponent, random oracle
   • Q: Can this be avoided?

3. Only public policies considered
   • Q: Can we achieve private policy?

4. Additional features
   • Easy: credential expiration, delegation, multi-authority, CA unlinkability
   • Q: Can we support credential revocation, CA-verifier collusion?
Summary

Introduced *attribute-based key exchange*
- Built from (1) zero-knowledge using garbled circuits
  (2) attribute selective encryption
- Very efficient implementation
- Lots of open problems

Interesting application of secure computation primitives/ideas to “other area” of cryptography
Thank you!

Full Version: https://eprint.iacr.org/2016/518
Code: https://www.github.com/amaloz/abke