Verifiable Computations from Garbled Circuit + FHE, & ABE

Presenter: LIU Yi

Paper

Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers

Rosario Gennaro[1], Craig Gentry[1], and Bryan Parno[2]

[1] IBM T.J.Watson Research Center
[2] CyLab, Carnegie Mellon University

CRYPTO 2010

Paper

How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption

Bryan Parno
Microsoft Research
parno@microsoft.com

Mariana Raykova
Columbia University
mariana@cs.columbia.edu

Vinod Vaikuntanathan*
University of Toronto
vinodv@cs.toronto.edu

TCC 2012

Paper

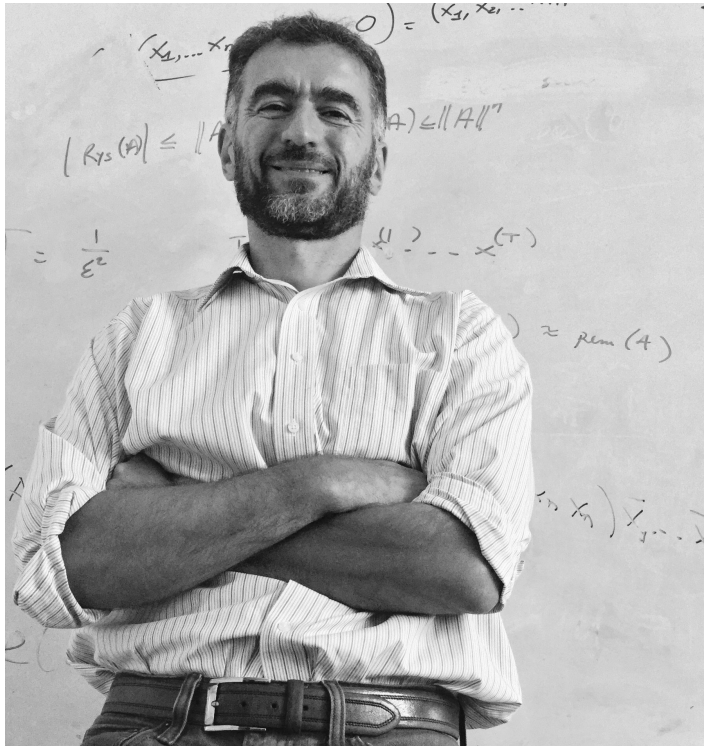
Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers

Rosario Gennaro[1], Craig Gentry[1], and Bryan Parno[2]

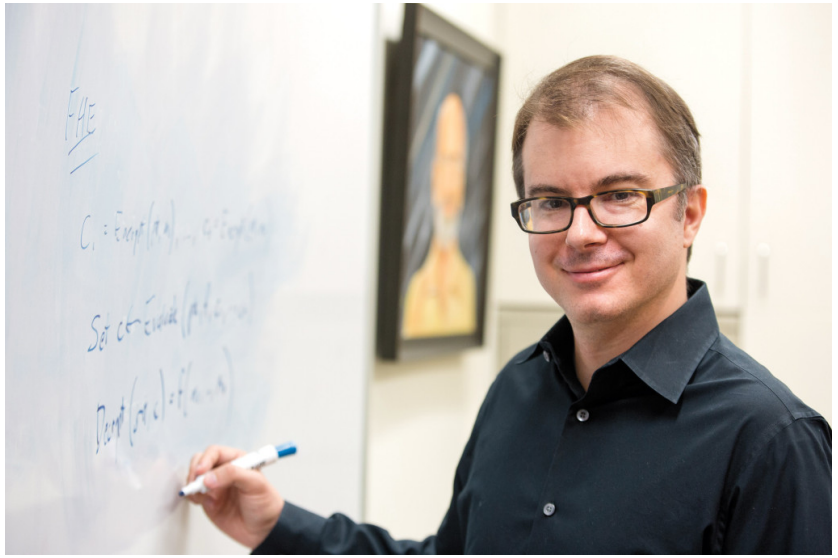
[1] IBM T.J.Watson Research Center
[2] CyLab, Carnegie Mellon University

CRYPTO 2010

Authors



Rosario Gennaro



Craig Gentry

Bryan Parno

Verifiable Computation Scheme

- This is a protocol between two polynomial-time parties, a *client* and a *worker*, to collaborate on the computation of a function $F: \{0,1\}^n \to \{0,1\}^m$
- The client can verify the correctness of output produced by the worker efficiently.

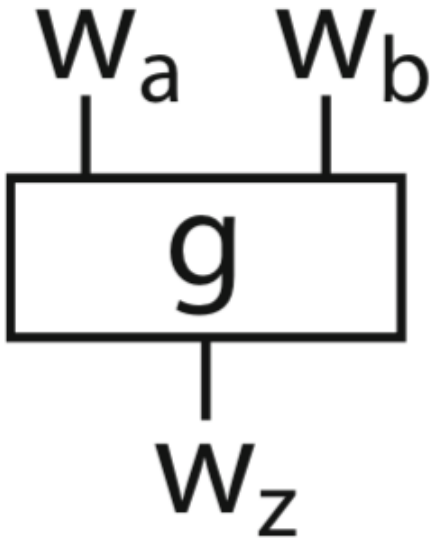
Verifiable Computation (in this paper)

- **Preprocessing.** A one-time stage in which the client computes some auxiliary (public and private) information associated with F. This phase can take time comparable to computing the function from scratch, but it is performed only once, and its cost is amortized over all the future executions.
- **Input Preparation.** When the client wants the worker to compute F(x), it prepares some auxiliary (public and private) information about x. The public information is sent to the worker.
- **Output Computation and Verification.** Once the worker has the public information associated with F and x, it computes a string πx which encodes the value F(x) and returns it to the client. From the value πx, the client can compute the value F(x) and verify its correctness.

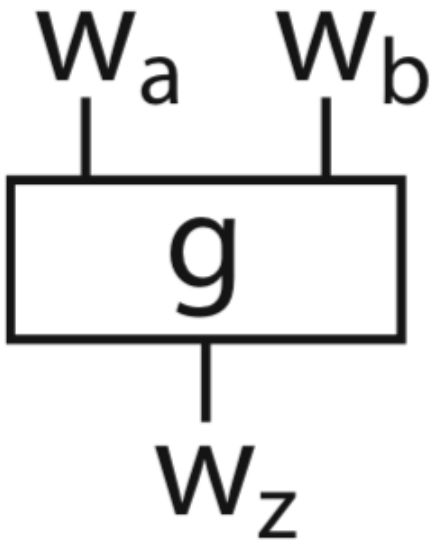
Additional Property

- Privacy. The construction has the added benefit of providing input and output privacy for the client, meaning that the worker does not learn any information about x or F(x).
  - This is bundled into the protocol and comes at no additional cost.
  - The work is the first to provide a weak client with the ability to efficiently and verifiably offload computation to an untrusted server in such a way that the input remains secret.

Yao’ s Garbled Circuit

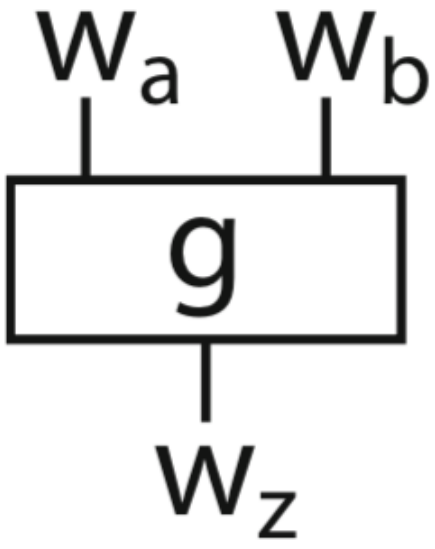


Yao’ s Garbled Circuit



w_a	w_b	w_z
0	0	g(0,0)
0	1	g(0,1)
1	0	g(1,0)
1	1	g(1,1)

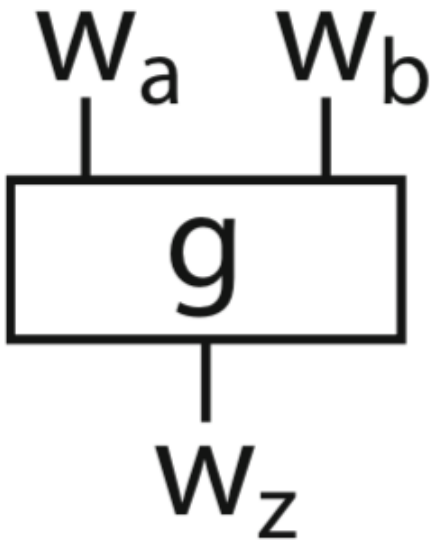
Yao’ s Garbled Circuit



w_a	w_b	w_z
0	0	g(0,0)
0	1	g(0,1)
1	0	g(1,0)
1	1	g(1,1)

w_a	w_b	w_z
k_a^0	k_b^0	$k_z^{g(0,0)}$
k_a^0	k_b^1	$k_z^{g(0,1)}$
k_a^1	k_b^0	$k_z^{g(1,0)}$
k_a^1	k_b^1	$k_z^{g(1,1)}$

Yao’ s Garbled Circuit

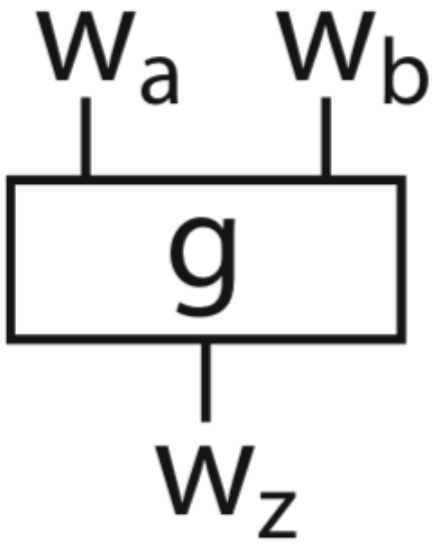


w_a	w_b	w_z
0	0	g(0,0)
0	1	g(0,1)
1	0	g(1,0)
1	1	g(1,1)

w_a	w_b	w_z
k_a^0	k_b^0	$k_z^{g(0,0)}$
k_a^0	k_b^1	$k_z^{g(0,1)}$
k_a^1	k_b^0	$k_z^{g(1,0)}$
k_a^1	k_b^1	$k_z^{g(1,1)}$

w_a	w_b	w_z
k_a^0	k_b^0	$E_{k_a^0}(E_{k_b^0}(k_z^{g(0,0)}))$
k_a^0	k_b^1	$E_{k_a^0}(E_{k_b^1}(k_z^{g(0,1)}))$
k_a^1	k_b^0	$E_{k_a^1}(E_{k_b^0}(k_z^{g(1,0)}))$
k_a^1	k_b^1	$E_{k_a^1}(E_{k_b^1}(k_z^{g(1,1)}))$

Yao’ s Garbled Circuit



w_a	w_b	w_z
k_a^0	k_b^0	$E_{k_a^0}(E_{k_b^0}(k_z^{g(0,0)}))$
k_a^0	k_b^1	$E_{k_a^0}(E_{k_b^1}(k_z^{g(0,1)}))$
k_a^1	k_b^0	$E_{k_a^1}(E_{k_b^0}(k_z^{g(1,0)}))$
k_a^1	k_b^1	$E_{k_a^1}(E_{k_b^1}(k_z^{g(1,1)}))$

Fully Homomorphic Encryption (DGHV scheme, from EUROCRYPT 10

- Enc(m):　m+2r+pq
- Dec(c):　(c mod p) mod 2=（c – p*「c/p」）mod 2 = Lsb(c) XOR Lsb(「c/p」)
- p是一个正的奇数，q是一个大的正整数（没有要求是奇数，它比p要大的多），p 和q在密钥生成阶段确定，p看成是密钥。而r是加密时随机选择的一个小的整数（可以为负数）
- 一个实数模p为：a mod p = a -「a/p」*p,「a」表示最近整数，即有唯一整数在（a-1/2, a+1/2]中。所以a mod p的范围也就变成了（-p/2，p/2]

Definition

A *verifiable computation scheme* $\mathcal{VC} = (\textbf{KeyGen}, \textbf{ProbGen}, \textbf{Compute}, \textbf{Verify})$ consists of the four algorithms defined below.

1. $\textbf{KeyGen}(F, \lambda) \rightarrow (PK, SK)$: Based on the security parameter λ, the randomized *key generation* algorithm generates a public key that encodes the target function F, which is used by the worker to compute F. It also computes a matching secret key, which is kept private by the client.
2. $\textbf{ProbGen}_{SK}(x) \rightarrow (\sigma_x, \tau_x)$: The *problem generation* algorithm uses the secret key SK to encode the function input x as a public value σ_x which is given to the worker to compute with, and a secret value τ_x which is kept private by the client.
3. $\textbf{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$: Using the client's public key and the encoded input, the worker *computes* an encoded version of the function's output $y = F(x)$.
4. $\textbf{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$: Using the secret key SK and the secret "decoding" τ_x, the *verification* algorithm converts the worker's encoded output into the output of the function, e.g., $y = F(x)$ or outputs $\perp$ indicating that σ_y does not represent the valid output of F on x.

Correctness

Definition 1 (Correctness). *A verifiable computation scheme $\mathcal{VC}$ is* correct *if for any function F, the key generation algorithm produces keys $(PK, SK) \leftarrow \mathbf{KeyGen}(F, \lambda)$ such that, $\forall x \in \mathtt{Domain}(F)$, if $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(x)$ and $\sigma_y \leftarrow \mathbf{Compute}_{PK}(\sigma_x)$ then $y = F(x) \leftarrow \mathbf{Verify}_{SK}(\tau_x, \sigma_y)$.*

Security

Experiment $\mathbf{Exp}_A^{Verif}[\mathcal{VC}, F, \lambda]$

$(PK, SK) \xleftarrow{R} \mathbf{KeyGen}(F, \lambda)$;

For $i = 1, \ldots, \ell = poly(\lambda)$;

$x_i \leftarrow A(PK, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;

$(\sigma_i, \tau_i) \leftarrow \mathbf{ProbGen}_{SK}(x_i)$;

$(i, \hat{\sigma}_y) \leftarrow A(PK, x_1, \sigma_1, \ldots, x_\ell, \sigma_\ell)$;

$\hat{y} \leftarrow \mathbf{Verify}_{SK}(\tau_i, \hat{\sigma}_y)$

If $\hat{y} \neq \perp$ and $\hat{y} \neq F(x_i)$, output '1', else '0';

Security

Definition 2 (Security). *For a verifiable computation scheme $\mathcal{VC}$, we define the advantage of an adversary A in the experiment above as:*

$$Adv_A^{Verif}(\mathcal{VC}, F, \lambda) = Prob[\mathbf{Exp}_A^{Verif}[\mathcal{VC}, F, \lambda] = 1] \tag{2}$$

A verifiable computation scheme $\mathcal{VC}$ is secure *for a function F, if for any adversary A running in probabilistic polynomial time,*

$$Adv_A^{Verif}(\mathcal{VC}, F, \lambda) \leq \texttt{negli}(\lambda) \tag{3}$$

where `negli`*() is a negligible function of its input.*

Input and Output Privacy

$$
\begin{aligned}
&\text{Experiment } \mathbf{Exp}_{A}^{Priv}[\mathcal{VC}, F, \lambda] \\
&\quad (PK, SK) \xleftarrow{R} \mathbf{KeyGen}(F, \lambda); \\
&\quad (x_0, x_1) \leftarrow A^{\mathbf{PubProbGen}_{SK}(\cdot)}(PK) \\
&\quad (\sigma_0, \tau_0) \leftarrow \mathbf{ProbGen}_{SK}(x_0); \\
&\quad (\sigma_1, \tau_1) \leftarrow \mathbf{ProbGen}_{SK}(x_1); \\
&\quad b \xleftarrow{R} \{0, 1\}; \\
&\quad \hat{b} \leftarrow A^{\mathbf{PubProbGen}_{SK}(\cdot)}(PK, x_0, x_1, \sigma_b) \\
&\quad \text{If } \hat{b} = b, \text{ output '1', else '0';}
\end{aligned}
$$

The oracle **PubProbGen***SK* (*x*) calls **ProbGen***SK* (*x*) to obtain (σ*x*,τ*x*) and returns only the public part σ*x*

Efficiency

Definition 4 (Outsourceable). *A $\mathcal{VC}$ can be outsourced if it permits efficient generation and efficient verification. This implies that for any x and any σ_y, the time required for* $\mathbf{ProbGen}_{SK}(x)$ *plus the time required for* $\mathbf{Verify}(\sigma_y)$ *is $o(T)$, where T is the fastest known time required to compute $F(x)$.*

Input and Output Privacy

Definition 3 (Privacy). *For a verifiable computation scheme $\mathcal{VC}$, we define the advantage of an adversary A in the experiment above as:*

$$Adv_A^{Priv}(\mathcal{VC}, F, \lambda) = \left| Prob[\mathbf{Exp}_A^{Priv}[\mathcal{VC}, F, \lambda] = 1] - \frac{1}{2} \right| \tag{4}$$

A verifiable computation scheme $\mathcal{VC}$ is private *for a function F, if for any adversary A running in probabilistic polynomial time,*

$$Adv_A^{Priv}(\mathcal{VC}, F, \lambda) \leq \texttt{negli}(\lambda) \tag{5}$$

where `negli`*() is a negligible function of its input.*

Protocol

1. **KeyGen**$(F, \lambda) \rightarrow (PK, SK)$: Represent F as a circuit C. Following Yao's Circuit Construction (see Section 2), choose two values, $w_i^0, w_i^1 \xleftarrow{R} \{0,1\}^\lambda$ for each wire w_i. For each gate g, compute the four ciphertexts $(\gamma_{00}^g, \gamma_{01}^g, \gamma_{10}^g, \gamma_{11}^g)$ described in Equation 1. The public key PK will be the full set of ciphertexts, i.e., $PK \leftarrow \cup_g(\gamma_{00}^g, \gamma_{01}^g, \gamma_{10}^g, \gamma_{11}^g)$, while the secret key will be the wire values chosen: $SK \leftarrow \cup_i(w_i^0, w_i^1)$.

Protocol

2. $\mathbf{ProbGen}_{SK}(x) \rightarrow \sigma_x$: Run the fully-homomorphic encryption scheme's key generation algorithm to create a new key pair: $(PK_{\mathcal{E}}, SK_{\mathcal{E}}) \leftarrow \mathbf{KeyGen}_{\mathcal{E}}(\lambda)$. Let $w_i \subset SK$ be the wire values representing the binary expression of x. Set the public value $\sigma_x \leftarrow (PK_{\mathcal{E}}, \mathbf{Encrypt}_{\mathcal{E}}(PK_{\mathcal{E}}, w_i))$ and the private value $\tau_x \leftarrow SK_{\mathcal{E}}$.

Protocol

3. **Compute**$_{PK}(\sigma_x) \rightarrow \sigma_y$: Calculate **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \gamma_i)$. Construct a circuit Δ that on input w, w', γ outputs $D_w(D_{w'}(\gamma))$, where D is the decryption algorithm corresponding to the encryption E used in Yao's garbling (therefore Δ computes the appropriate decryption in Yao's construction). Calculate **Evaluate**$_{\mathcal{E}}(\Delta,$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, w_i),$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \gamma_i))$ repeatedly, to decrypt your way through the ciphertexts, just as in the evaluation of Yao's garbled circuit. The result is $\sigma_y \leftarrow$ **Encrypt**$_{\mathcal{E}}(PK_{\mathcal{E}}, \bar{w}_i)$, where $\bar{w}_i$ are the wire values representing $y = F(x)$ in binary.

Protocol

4. $\textbf{Verify}_{SK}(\sigma_y) \rightarrow y \cup \perp$: Use $SK_{\mathcal{E}}$ to decrypt $\textbf{Encrypt}_{\mathcal{E}}(PK_{\mathcal{E}}, \bar{w}_i)$, obtaining $\bar{w}_i$. Use SK to map the wire values to an output y. If the decryption or mapping fails, then output $\perp$.

Verifying ciphertext ranges in an encrypted form

$$\mathbf{Encrypt}_{\mathcal{E}}(PK_{\mathcal{E}}, \Sigma_i M(k, \gamma_i) \bar{D}_k(\bar{\gamma}_i))$$

Cheating Workers

- The worker want to determine the value of a label w^b
- Assume that the encryption is bit level, $b = b_1 b_2 \cdots b_n$, $Enc(b) = Enc(b_1)||Enc(b_2) \cdots ||Enc(b_n)$
- Compute the correct output as $Enc(O)$
- XOR $Enc(b_1)$ with $Enc(O_1)$ using FHE properties and send this $Enc(O')$ to the client.
- If the client accept, then $b_1 = 0$, otherwise, $b_1 = 1$.

Paper

How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption

Bryan Parno
Microsoft Research
parno@microsoft.com

Mariana Raykova
Columbia University
mariana@cs.columbia.edu

Vinod Vaikuntanathan*
University of Toronto
vinodv@cs.toronto.edu

TCC 2012

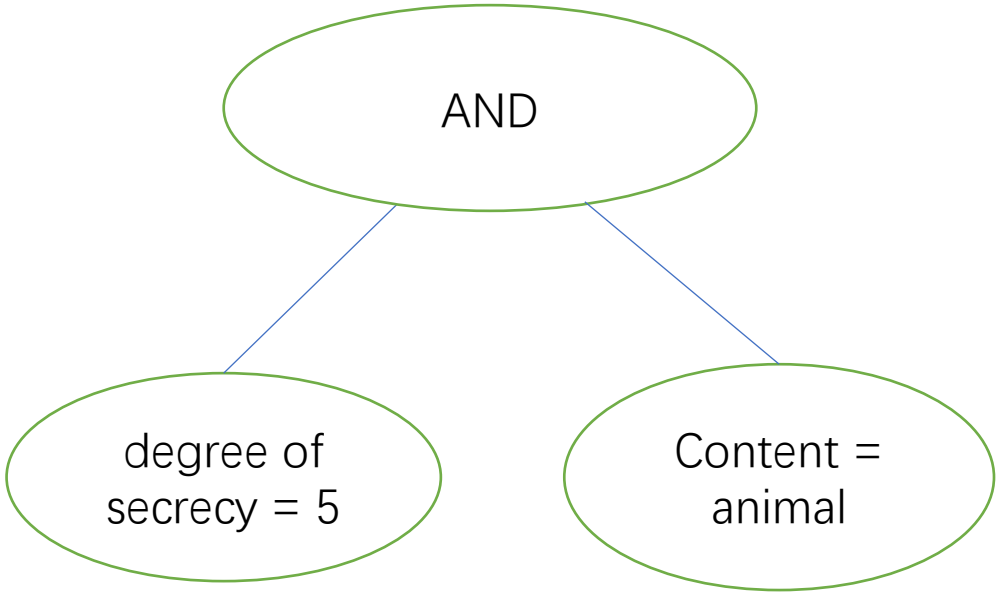
Authors

Bryan Parno

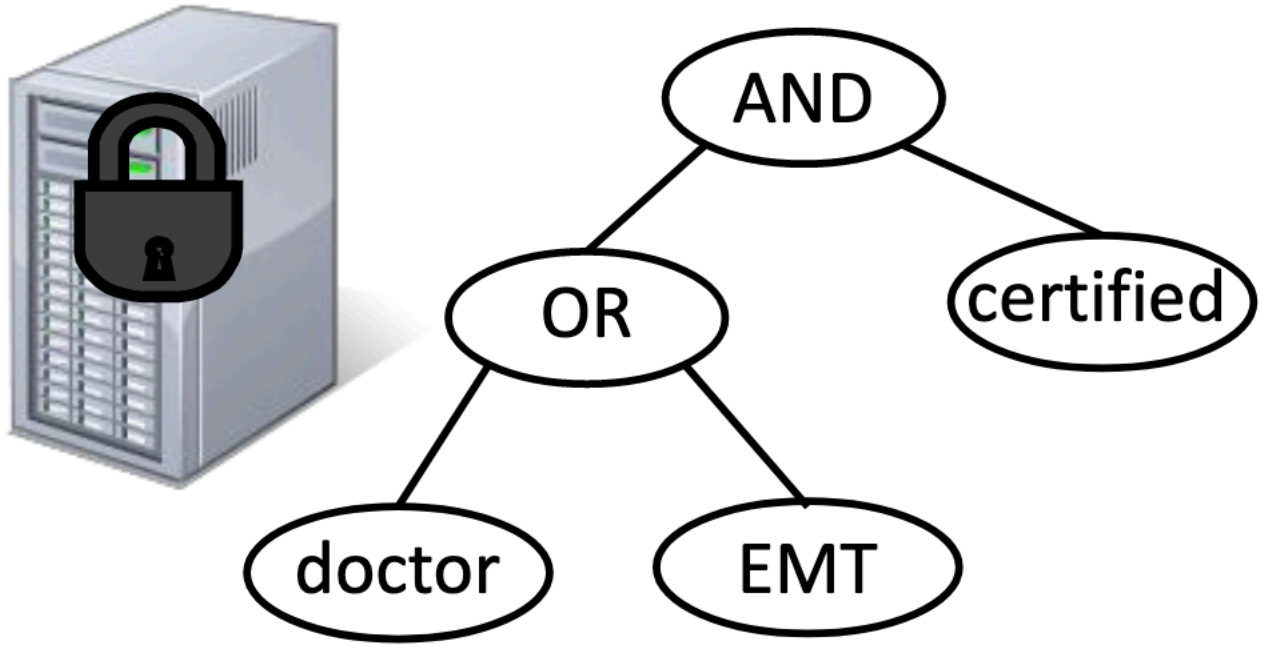
Mariana Raykova

Vinod Vaikuntanathan

Key-Policy and Ciphertext-Policy



KP



CP

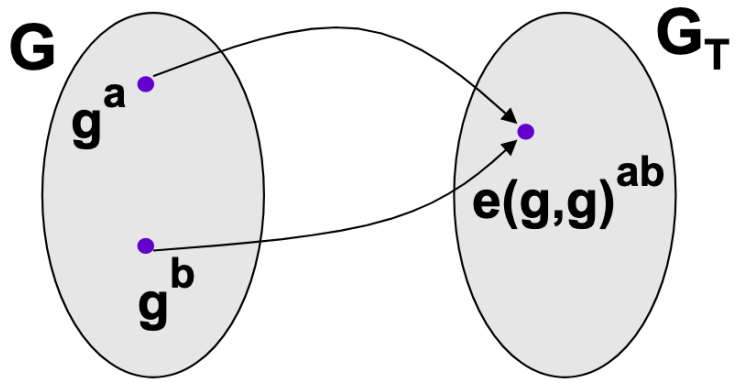
KP-ABE

Definition 5 (Key-Policy Attribute-Based Encryption) *An attribute-based encryption scheme $\mathcal{ABE}$ is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *defined as follows:*

- $\mathsf{Setup}(\lambda, U) \rightarrow (PK, MSK)$ *: Given a security parameter* λ *and the set of all possible attributes* U*, output a public key* PK *and a master secret key* MSK*.*
- $\mathsf{Enc}_{PK}(M, \gamma) \rightarrow C$*: Given a public key* PK*, a message* M*, and a set of attributes* γ*, output ciphertext* C*.*
- $\mathsf{KeyGen}_{MSK}(F) \rightarrow SK_F$*: Given a function* F *and the master secret key* MSK*, output a decryption key* SK_F *associated with that function.*
- $\mathsf{Dec}_{SK_F}(C) \rightarrow M \cup \perp$*: Given a ciphertext* $C = \mathsf{Enc}_{PK}(M, \gamma)$ *and a secret key* SK_F *for function* F*, output* M *if* $F(\gamma) = 1$*, or* $\perp$*, otherwise.*

Bilinear Map

More abstractly: bilinear groups



G , G_T : finite cyclic groups of prime order q.

- Def: A **pairing** $e: G \times G \rightarrow G_T$ is a map:
  - Bilinear: $e(g^a, g^b) = e(g,g)^{ab}$ $\forall a,b \in Z,\ g \in G$
  - Poly-time computable and non-degenerate:

    g generates G $\Rightarrow$ e(g,g) generates G_T

Exercises

Reduce $e(g^a, h)e(g^b, h)$, $e(g, h^a)e(g^b, h)$, $e(g^a, h^{-b})e(u, v)e(f, h)^c$, $\prod_{i=1}^{n} e(g, h^{a_i})^{b_i}$

Reduce $e(u, v)e(u, w)$, $e(u, v^a)e(u^b, v)$, $e(g^a, v^{-b})e(f, w)e(u, v)^c$, $\prod_{i=1}^{n} e(u^a, v_i^b)^{\frac{c_i}{ab}}$

Exercises

Reduce $e(g^a,h)e(g^b,h)$, $e(g,h^a)e(g^b,h)$, $e(g^a,h^{-b})e(u,v)e(g,h)^c$, $\prod_{i=1}^{n} e(g,h^{a_i})^{b_i}$

$$e(g^a,h)e(g^b,h) = e(g,h)^a e(g,h)^b = e(g,h)^{a+b}$$
$$e(g,h^a)e(g^b,h) = e(g,h)^a e(g,h)^b = e(g,h)^{a+b}$$
$$e(g^a,h^{-b})e(u,v)e(g,h)^c = e(g,h)^{-ab} e(g,h)^c e(u,v) = e(g,h)^{c-ab} e(u,v)$$
$$\prod_{i=1}^{n} e(g,h^{a_i})^{b_i} = \prod_{i=1}^{n} e(g,h)^{a_i b_i} = e(g,h)^{\sum_{i=1}^{n} a_i b_i}$$

Bilinear: For all $a,b \in \boldsymbol{Z}_p$: $e(g^a,h^b) = e(g,h)^{ab}$

Exercises

Reduce $e(u,v)e(u,w)$, $e(u,v^a)e(u^b,v)$, $e(g^a,v^{-b})e(f,w)e(u,v)^c$, $\prod_{i=1}^{n} e(u^a,v_i^b)^{\frac{c_i}{ab}}$

Because g generates G_1 we can write any $u \in G_1$ as $u = g^x$
Similarly, we can write any $v, w \in G_2$ as $v = h^y$ and $w = h^z$
- All we know is such $x, y, z \in \boldsymbol{Z}_p$ exist, we may not know what they are

$$e(u,v)e(u,w) = e(g^x,h^y)e(g^x,h^z) = e(g,h)^{x(y+z)} = e(u,vw)$$
$$e(u,v^a)e(u^b,v) = e(u,v)^a e(u,v)^b = e(u,v)^{a+b}$$
$$e(g^a,v^{-b})e(f,w)e(u,v)^c = e(g,v)^{-ab}e(g^x,v)^c e(f,w) = e(g^{-ab}u^c,v)e(f,w)$$
$$\prod_{i=1}^{n} e(u^a,v_i^b)^{\frac{c_i}{ab}} = \prod_{i=1}^{n} e(u,v_i)^{ab\cdot\frac{c_i}{ab}} = \prod_{i=1}^{n} e(u,v_i)^{c_i} = e\left(u,\prod_{i=1}^{n} v_i^{c_i}\right)$$

Bilinear: For all $a, b \in \boldsymbol{Z}_p$: $e(g^a,h^b) = e(g,h)^{ab}$

Simplest example: BLS signatures

[B-Lynn-Shacham'01]

KeyGen: sk = rand. x in Z_q , pk = $g^x \in G$

Sign(sk, m) $\longrightarrow$ $H(m)^x \in G$

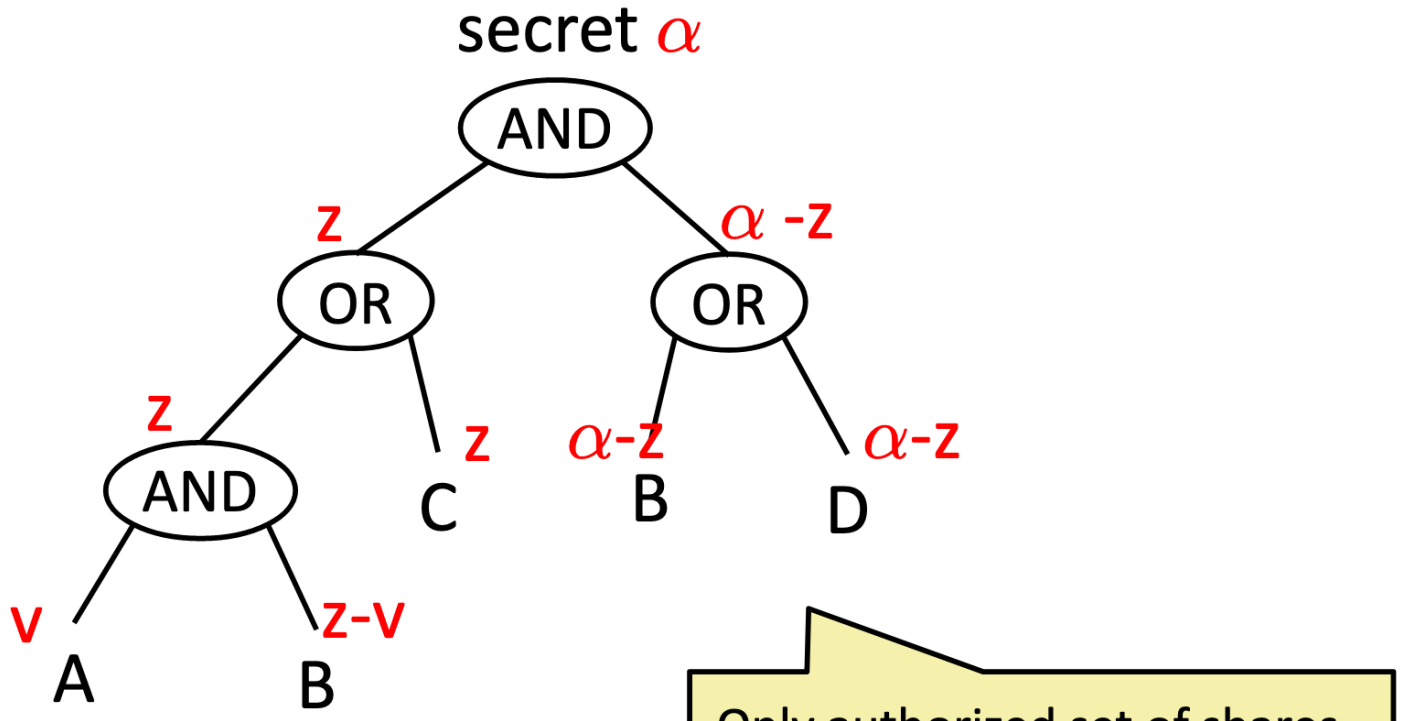
Verify(pk, m, sig) $\longrightarrow$ accept iff $e(g, H(m)^x) = e(g^x, H(m))$

$e(g, \text{sig}) \stackrel{?}{=} e(pk, H(m))$

Thm: Existentially unforgeable under CDH in the RO model

Construction Tools

Linear Secret Sharing:



Only authorized set of shares allows reconstruction of α

Construction Tools

Canceling with independent randomness on two sides:

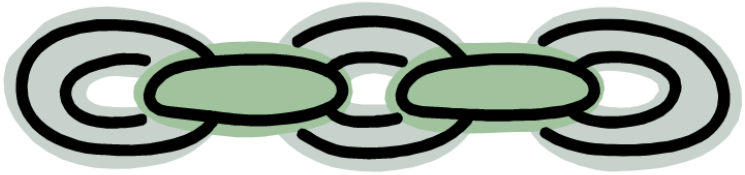
example:

$$K_1 = g^{ar}, \quad K_2 = g^{r}$$

$$C_1 = g^{-s}, \quad C_2 = g^{as}$$

Independent randomness

$$e(C_1, K_1)e(C_2, K_2) = e(g, g)^{-sar}e(g, g)^{sar} = 1$$



links computations together

Basic KP-ABE Construction[GPSW06]

Setup:

bilinear group G of prime order p, generator g
random exponent $\alpha \in \mathbb{Z}_p$,
random elements $H_i \in G$ for each $i \in U$

$$PP := \{g, e(g,g)^{\alpha},\ H_i\ \forall i \in U\} \quad MSK := \alpha$$

KeyGen(f):

split α into shares $\{\lambda_i\}$ following f
choose random $r_i \in \mathbb{Z}_p$

personalized randomness

$$SK = \{g^{\lambda_i} H_i^{r_i},\ \ g^{r_i}\}$$

Encrypt$(M, S \subseteq U)$:

choose random $s \in \mathbb{Z}_p$

$$CT = Me(g,g)^{\alpha s},\ g^s,\ \{H_i^s\}_{i \in S}$$

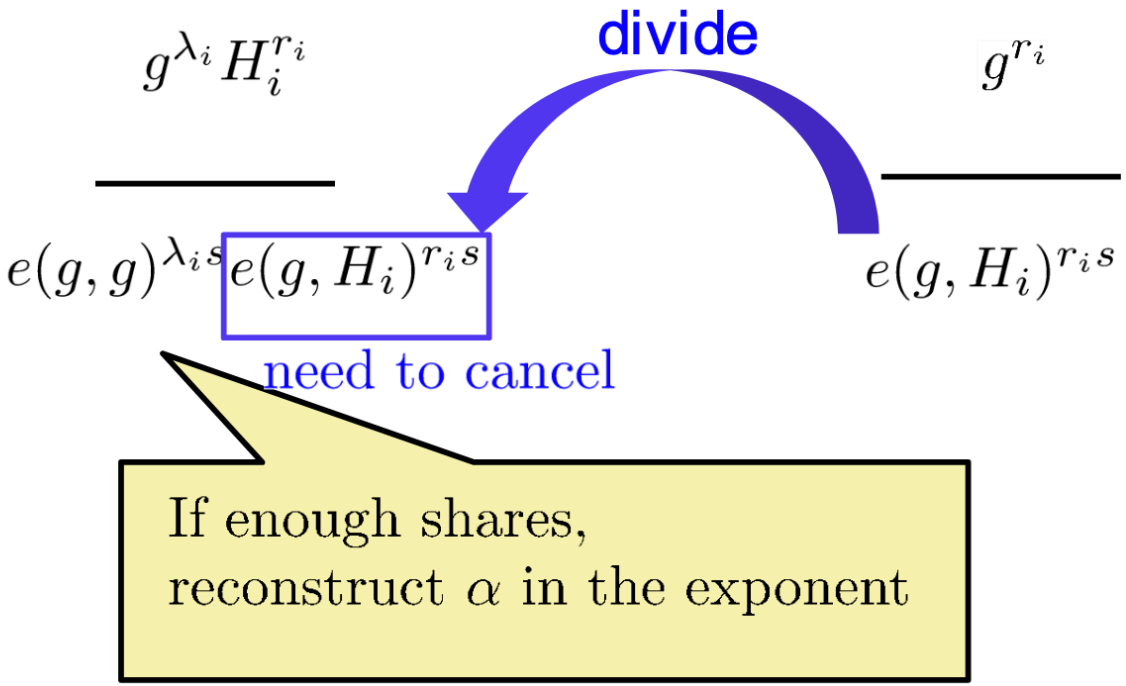
Decryption

Goal: recover M

We have: $Me(g,g)^{\alpha s}$ ⟹ Subgoal: compute $e(g,g)^{\alpha s}$

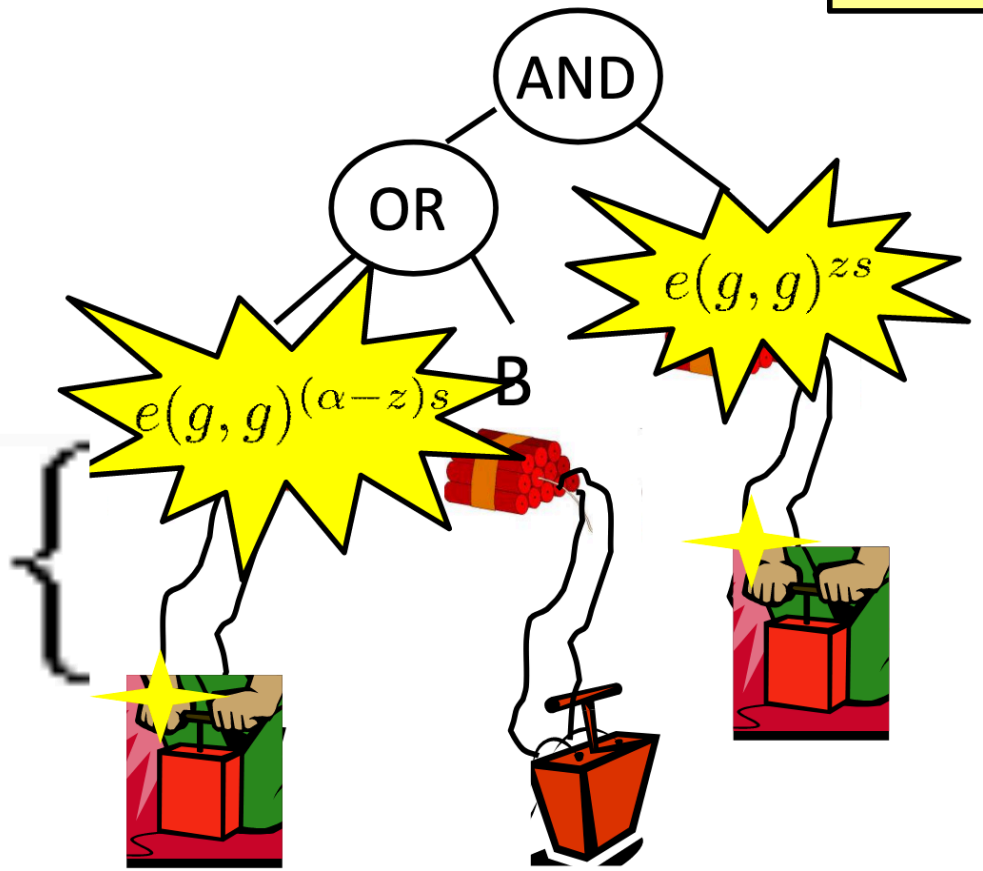
CT: g^s H_i^s

SK: $g^{\lambda_i} H_i^{r_i}$ g^{r_i}



High Level View of Scheme

$e(g,g)^{(\alpha-z)s}e(g,g)^{zs}$ blinding factor



Chain Reaction

A in CT

C in CT

From ABE to VC

- Policy $f(\cdot)$
- Attribute x
- If $f(x) = 1$, we can decrypt the ciphertext, otherwise, we cannot.

- Delegated function f, worker is given $\mathbf{sk_f}$, we encrypt message m using attribute x
- If worker successfully returns m, we know $f(x) = 1$
- What about $f(x) = 0$?
  - Define $\bar{f}$

Acknowledgement

- Some materials are extracted from the slides created by Prof. Dan Boneh, Jens Groth, and Allison Lewko.