Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation

Manuel Barbosa[1] **Pooya Farshim**[2]

[1]Universidade do Minho, Portugal

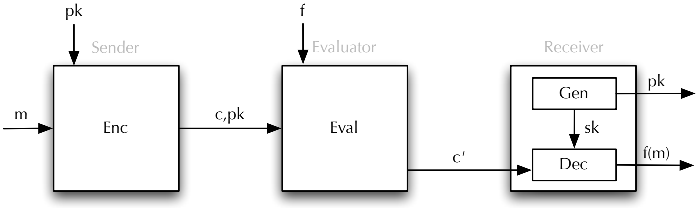
[2]**Technischen Universität Darmstadt, Germany**

CT-RSA 2012
01.03.2012

Background

Fully Homomorphic Encryption

Allows computing over encrypted data:

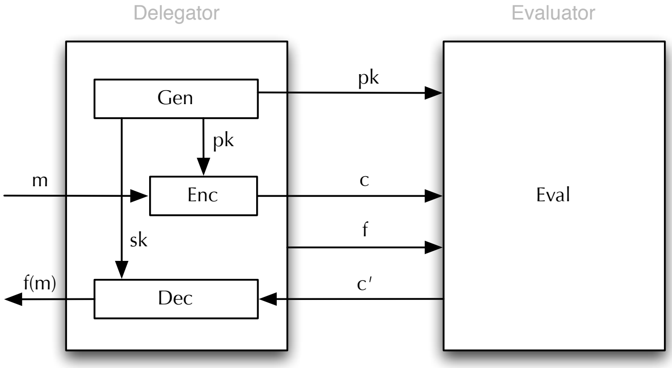


$$
\begin{aligned}
(\mathsf{sk}, \mathsf{pk}) &\leftarrow_{\$} \mathsf{Gen}(1^\lambda) \\
\mathsf{c} &\leftarrow_{\$} \mathsf{Enc}(\mathsf{m}, \mathsf{pk}) \\
\mathsf{c}' &\leftarrow_{\$} \mathsf{Eval}(\mathsf{c}, f, \mathsf{pk}) \\
f(\mathsf{m}) &= \mathsf{Dec}(\mathsf{c}', \mathsf{sk})
\end{aligned}
$$

Security: Standard IND-CPA security.

Fully Homomorphic Encryption

Can privately outsource computation:



FHE compact ⇒ protocol outsourcing

Verifiable Computation

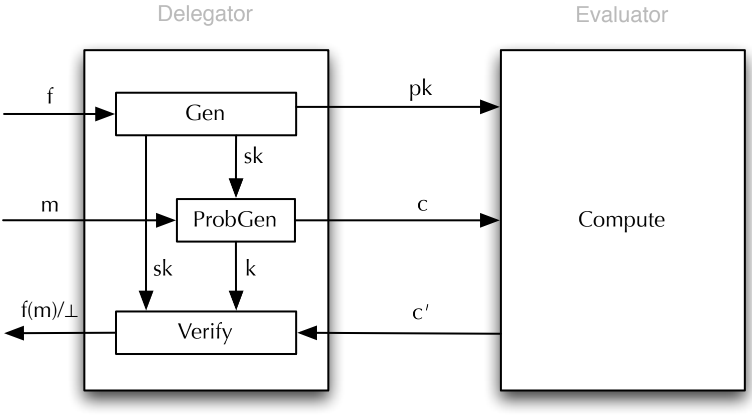
FHE-based solution is not verifiable:

Evaluator may compute $\tilde{f}$ instead of f.

A verifiable computation (VC) scheme allows **verifiable** outsourcing of computation:

$$\begin{aligned}
(\mathsf{sk}, \mathsf{pk}) &\leftarrow_{\$} \mathsf{Gen}(f, 1^{\lambda}) \\
(\mathsf{c}, \mathsf{k}) &\leftarrow_{\$} \mathsf{ProbGen}(\mathsf{m}, \mathsf{sk}) \\
\mathsf{c}' &\leftarrow_{\$} \mathsf{Compute}(\mathsf{c}, \mathsf{pk}) \\
f(\mathsf{m}) \text{ or } \perp &= \mathsf{Verify}(\mathsf{c}', \mathsf{k}, \mathsf{sk})
\end{aligned}$$

Verifiable Oustsourcing of Computation



$$\text{Time}(\text{Gen}) = O(f) \quad \text{and} \quad \text{Time}(\text{Verify}) = o(f)$$

Security: Input/Output (I/O) Privacy

No information about the input (and hence the output) is leaked.

proc. Initialize(f, λ):
$b \leftarrow_{\$} \{0,1\}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow_{\$} \mathsf{Gen}(f, 1^\lambda)$
Return pk

proc. PubProbGen(m):
$(\mathsf{c}, \mathsf{k}) \leftarrow_{\$} \mathsf{ProbGen}(\mathsf{m}, \mathsf{sk})$
Return c

proc. LR$(\mathsf{m}_0, \mathsf{m}_1)$:
$\mathsf{c} \leftarrow_{\$} \mathsf{ProbGen}(\mathsf{m}_b, \mathsf{sk})$
Return c

proc. Finalize(b'):
Return $(b = b')$

$$\mathbf{Adv}^{\text{ind-cpa}}_{f,\mathsf{VC},\mathcal{A}}(\lambda) := 2 \cdot \Pr\left[\mathsf{Game}^{\mathcal{A}} \Rightarrow \mathsf{T}\right] - 1$$

Security: Verifiability

Adversary cannot fool the delegator to accept a wrong result.

proc. Initialize(f, λ):
List $\leftarrow \{\}$; $i \leftarrow 0$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow_{\$} \mathsf{Gen}(f, 1^{\lambda})$
Return pk

proc. PubProbGen(m):
$(\mathsf{c}, \mathsf{k}) \leftarrow_{\$} \mathsf{ProbGen}(\mathsf{m}, \mathsf{sk})$
$i \leftarrow i + 1$
List $\leftarrow$ List $\cup \{(i, \mathsf{m}, \mathsf{k})\}$
Return c

proc. PubVerify(c, i):
Find (m, k) s.t. $(i, \mathsf{m}, \mathsf{k}) \in$ List
$\mathsf{m} \leftarrow \mathsf{Verify}(\mathsf{c}, \mathsf{k}, \mathsf{sk})$
Return m

proc. Finalize$(\mathsf{c}^\star, i)$:
If $(i, \star, \star) \notin$ List Return F
Find (m, k) s.t. $(i, \mathsf{m}, \mathsf{k}) \in$ List
$\mathsf{m}^\star \leftarrow \mathsf{Verify}(\mathsf{c}^\star, \mathsf{k}, \mathsf{sk})$
Return $(\mathsf{m}^\star \neq \perp \wedge \mathsf{m}^\star \neq f(\mathsf{m}))$

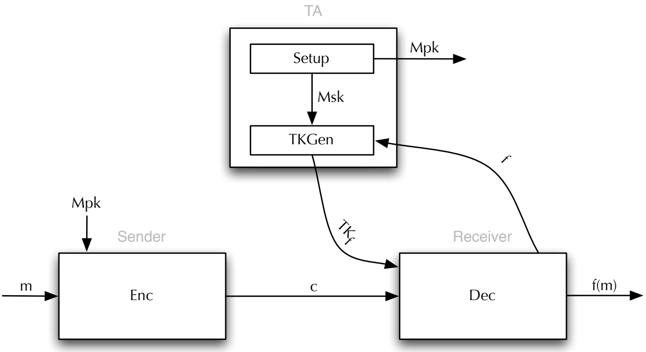
$$\mathbf{Adv}^{\text{vrf-ccax}}_{f,\mathsf{VC},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Game}^{\mathcal{A}} \Rightarrow \mathsf{T}\right]$$

(Non-interactive) Outsourcing of Computation

- Prior work:
  - Literature from complexity theory: PCPs + CS proofs, where verifier checks a small/const number of bits of the proof.
  - Yao's garbled circuit + FHE [GGP10].
  - Cut-and-choose protocol + FHE [CKV10].
  - These schemes are not fully verifiable.
- Large body of recent work on related topics:
  - Verifiable Computation with Two or More Clouds, CCS 2011.
  - Outsourcing the Decryption of ABE Ciphertexts, Usenix 2011.
  - How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption, TCC 2012.
  - Delegation of Computation without Rejection Problem from Designated Verifier CS-proofs, ePrint 2011.
  - Targeted Malleability: Homomorphic Encryption for Restricted Computations, ITCS 2012.

  ...

Functional Encryption



$$
\begin{aligned}
(\mathsf{Msk}, \mathsf{Mpk}) &\leftarrow_{\$} \mathsf{Setup}(1^{\lambda}) \\
\mathsf{TK}_f &\leftarrow_{\$} \mathsf{TKGen}(f, \mathsf{Msk}) \\
\mathsf{c} &\leftarrow_{\$} \mathsf{Enc}(\mathsf{m}, \mathsf{Mpk}) \\
f(\mathsf{m}) / \perp &= \mathsf{Dec}(\mathsf{c}, \mathsf{TK}_f)
\end{aligned}
$$

Generalizes many primitives such as: PKE, IBE, ABE, PE, . . .

Security: Indistinguishability

proc. Initialize(λ):

$b \leftarrow_\$ \{0,1\}$
$(\mathsf{Msk}, \mathsf{Mpk}) \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
Return Mpk

oracle LR($\mathsf{m}_0, \mathsf{m}_1$):

$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}_b, \mathsf{Mpk})$
Return c

oracle Token(f):

$\mathsf{TK} \leftarrow_\$ \mathsf{TKGen}(f, \mathsf{Msk})$
$\mathsf{TKList} \leftarrow f : \mathsf{TKList}$
Return TK

proc. Finalize(b'):

Return $(b = b')$

An adversary is legitimate if:

- $\mathsf{R}(\mathsf{m}_0, \mathsf{m}_1) = 1$. Typically $\mathsf{R}(\mathsf{m}_0, \mathsf{m}_1) := (|\mathsf{m}_0| = |\mathsf{m}_1|)$.
- For all $f \in \mathsf{TKList}$ we have $f(\mathsf{m}_0) = f(\mathsf{m}_1)$.
- TNA model: it does not call **Token** after calling **LR**.

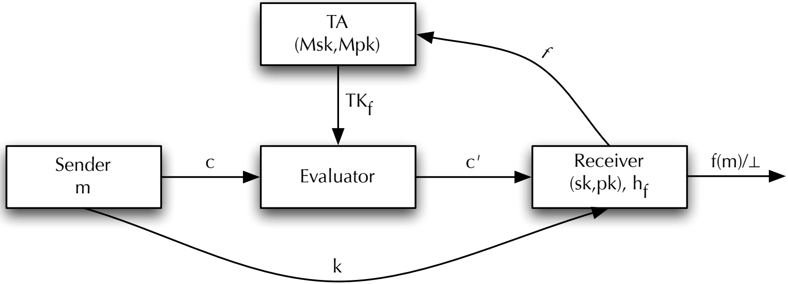
CCA1/2 model: add a **Decrypt** oracle.

Limitations of Known Primitives

- Fully Homomorphic Encryption (FHE):
  - Unrestricted evaluation.
  - No verifiability.
- Functional Encryption (FE):
  - No output privacy (for outsourcing).
  - No verifiability.
- Verifiable computation (VC):
  - Gen, ProbGen, and Verifier are the same party.
  - Support for a single function only.
  - (Until now) Not fully verifiable.

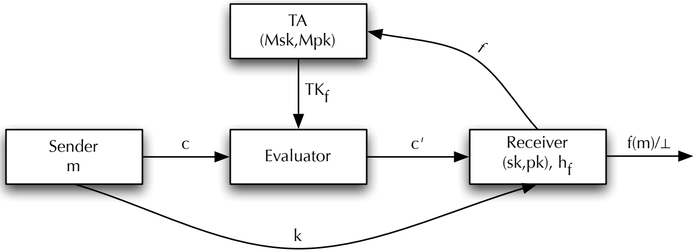
Delegatable Homomorphic Encryption

New Architecture



- Sender, Receiver, TA, and Evaluator have separate roles.
- Encryption is a public operation.
- One-time setup procedure for all *f*.
- k binds the computation to a specific *m*.
- h_f binds the computation to a specific *f*.
- I/O privacy, verifiability, and collusion resistance.

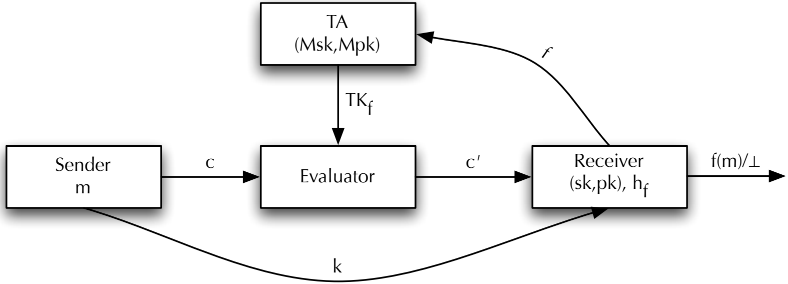
Examples



Health Record Statistics:

- Alice (Sender) has encrypted health records.
- Bob (Receiver) likes to obtain some statistics.
- Neither Alice nor Bob have enough computational resources.
- Carol (Evaluator) will compute over data.
- TA issues tokens so Carol computes the specific statistics (can even sell statistics).
- Bob is assured that I/O remain private, and the result is correct.

Examples



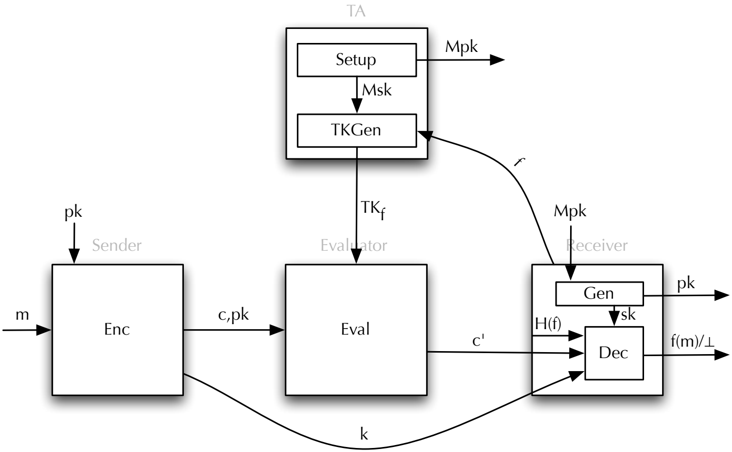
Email Filtering:

- Alice (Sender) sends encrypted emails to Bob (Receiver).
- Bob would like to filter emails.
- Bob does not have enough computational resources.
- TA issues token so Carol can run the specific filtering procedure.
- Carol (Evaluator) will filter emails for Bob.
- Bob is assured nothing is leaked, and filtering is done properly.

The DHE Primitive

$$
\begin{aligned}
(\mathsf{Msk}, \mathsf{Mpk}) &\leftarrow_\$ \mathsf{Setup}(1^\lambda) \\
(\mathsf{sk}, \mathsf{pk}) &\leftarrow_\$ \mathsf{Gen}(\mathsf{Mpk}) \\
(\mathsf{TK}_f, \mathsf{h}_f) &\leftarrow_\$ \mathsf{TKGen}(f, \mathsf{Msk}) \\
(\mathsf{c}, \mathsf{k}) &\leftarrow_\$ \mathsf{Enc}(\mathsf{m}, \mathsf{pk}) \\
\mathsf{c}' &\leftarrow_\$ \mathsf{Eval}(\mathsf{c}, \mathsf{TK}_f, \mathsf{pk}) \\
f(\mathsf{m}) \text{ or } \perp &= \mathsf{Dec}(\mathsf{c}', \mathsf{k}, \mathsf{h}_f, \mathsf{sk})
\end{aligned}
$$

The DHE Primitive



- A public-key counterpart to VC.
- Provides “targeted malleability".
- FHE where homomorphisms are delegated.

Security

Three notions:

I/O Privacy No information leaks about the data, even given the Msk and k.
(No access to a Verification oracle.)

Verifiability Adversary cannot fool the delegator to accept a wrong result.

Collusion Resistance Adversary knowing receiver's secret key cannot learn more than the result of the computations.

The Construction

Adding Verifiability to Functions

Given a function f, transform it to a function $f^\star$ by setting:

$$f^\star(\mathsf{m}, \mathsf{k}) := (f(\mathsf{m}), \mathsf{MAC}(f(\mathsf{m}) | \mathsf{h}_f, \mathsf{k}, \mathsf{mk}))\,.$$

Here

$$\mathsf{h}_f \leftarrow \mathsf{H}_{\mathsf{hk}}(\langle f \rangle)$$

where H is a collision-resistant hash function.

The Construction

- Transform f to $f^\star$ as above.
- Tokens are for the transformed functions.
- Encrypt functionally and then homomorphically.
- To evaluate, homomorphically functionally decrypt.
- To recover the result decrypt, and then verify the MAC.
- Use the function fingerprint and the auxiliary info for this.

n-Key-Chameleon MAC

Need a special MAC for the security proof:

$$\begin{aligned}(\mathsf{td}, \mathsf{mk}) &\leftarrow_{\$} \mathsf{Setup}(1^\lambda)\\ \mathsf{tag} &\leftarrow_{\$} \mathsf{MAC}(\mathsf{m}, \mathsf{k}, \mathsf{mk})\\ \mathsf{k}' &\leftarrow_{\$} \mathsf{Col}(\mathsf{td}, \mathsf{m}_1, \ldots, \mathsf{m}_n, \mathsf{k}, \mathsf{mk})\end{aligned}$$

For all m_i, must have:

$$\mathsf{MAC}(\mathsf{m}_i, \mathsf{k}, \mathsf{mk}) = \mathsf{MAC}(\mathsf{m}_i, \mathsf{k}', \mathsf{mk})$$

Security: $(n+1)$-time unforgeable when given k'.
Construction:

$$\mathsf{MAC}(\mathsf{m}, \underbrace{(a_n, \ldots, a_0)}_{k}, \epsilon) := \sum_{i=0}^{n} a_i \mathsf{m}^i$$

Collision: solve n equations in $n+1$ unknowns.

Security Guarantees

Theorem

The DHE construction provides input/output privacy, verifiability, and collusion resistance if the FE scheme is IND-CCA1, the FHE is IND-CPA, and the MAC is unforgeable.

$$\mathbf{Adv}^{\text{ta-ind-cpa}}_{\text{DHE},\mathcal{A}}(\lambda) = \mathbf{Adv}^{\text{ind-cpa}}_{\text{FHE},\mathcal{B}}(\lambda)$$

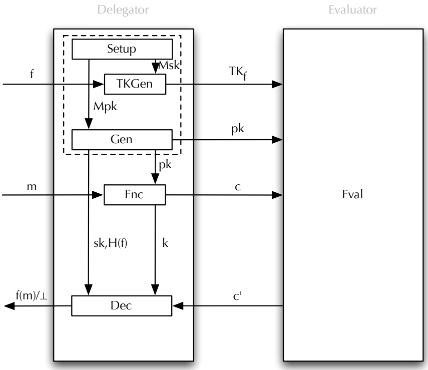
$$\mathbf{Adv}^{\text{ind-evalx}}_{\text{DHE},\mathcal{A}}(\lambda) = \mathbf{Adv}^{\text{ind-ccax}}_{\text{FE},\mathcal{B}}(\lambda)$$

$$\mathbf{Adv}^{\text{vrf-cca1}}_{\text{DHE},\mathcal{A}}(\lambda) \leq (\mathbf{Q}^{\textbf{Decrypt}}_{\text{DHE},\mathcal{A}}(\lambda) + 1) \cdot \mathbf{Q}^{\textbf{Encrypt}}_{\text{DHE},\mathcal{A}}(\lambda) \cdot (\mathbf{Adv}^{\text{ind-cca1}}_{\text{FE},\mathcal{B}}(\lambda) + \mathbf{Adv}^{\text{uf-cma}}_{\text{MAC},\mathcal{C}}(\lambda))$$

Proof

- I/O privacy follows from the security of the FHE layer.
- Collusion resistance follows from FE security.
- Verifiability:
  - $\mathbf{Q}^{\mathbf{Encrypt}}$: Adversary wins for the *i*-th encryption only.
  - $\mathbf{Q}^{\mathbf{Decrypt}} + 1$: The adversary is playing the game $\mathbf{Q}^{\mathbf{Decrypt}} + 1$ times: the $\mathbf{Q}^{\mathbf{Decrypt}}$ decrypt queries are answered with $\perp$.
  - *n*-Key-Chameleon property:
    - Change key from real to one generated through the collision algorithm.
    - $f^{\star}(\mathsf{m}, \mathsf{k}) = f^{\star}(\mathsf{m}, \mathsf{k}')$ due to the chameleon property (and legitimacy of the adversary).
    - Negligible hop down to IND-CCA1 security of FE.
  - Now reduce to the unforgeability of MAC. Note we have k' from MAC game.

DHE ⇒ VC



- VC.Gen: Run DHE.Setup + DHE.Gen + DHE.TKGen. Return $((h_f, sk, pk), (TK_f, pk))$.
- VC.ProbGen: Run DHE.Enc. Return (c, k).
- VC.Compute: Run DHE.Eval. Return c'.
- VC.Verify: Run DHE.Dec. Return y or $\perp$.

Further Research

Security:

- I/O privacy in the presence of a verification oracle.
  - The construction is insecure in this model: Change one bit at a time and then check it using the verification oracle.
- Unbounded/adaptive token queries.

DHE already quite powerful, but:

- Public verifiability.
- Multi/i-hop and multi-arity variants.
- Multiple evaluators with t out of n being honest.
- Randomized functions.

Also:

- Instantiations for specific functionalities (DHE & VFE).

By mixing homomorphic and functional encryption
and a special MAC
once can build a powerful variant of VC

Thank you for your attention.

Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting

Carmit Hazay[1] Gert Læssøe Mikkelsen[2]
Tal Rabin[3] Tomas Toft[1]

[1] Department of Computer Science, Aarhus University.
[2] The Alexandra Institute.
[3] IBM T.J.Watson Research Center.

March 1, 2012

ALEXANDRA INSTITUTE

Contributions:

1. Efficient Distributed RSA Moduli Generation
2. Threshold Paillier Encryption

Setting:

- Both in the Two-Party setting
- Security against active adversaries.
- Security proofs based on simulation.

Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p and q are primes)
- Generate p and q using the Miller-Rabin test
- Used in:
  - Encryption schemes
  - Signature schemes
  - Lots of other cryptographic tools
- Paillier Encrypion Scheme

Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p and q are primes)
- Generate p and q using the Miller-Rabin test
- Used in:
  - Encryption schemes
  - Signature schemes
  - Lots of other cryptographic tools
- Paillier Encrypion Scheme

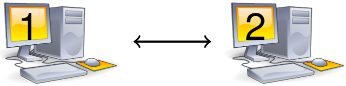
Distributed Generation

Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p and q are primes)
- Generate p and q using the Miller-Rabin test
- Used in:
  - Encryption schemes
  - Signature schemes
  - Lots of other cryptographic tools
- Paillier Encrypion Scheme

Distributed Generation

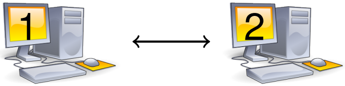


Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p and q are primes)
- Generate p and q using the Miller-Rabin test
- Used in:
  - Encryption schemes
  - Signature schemes
  - Lots of other cryptographic tools
- Paillier Encrypion Scheme

Distributed Generation



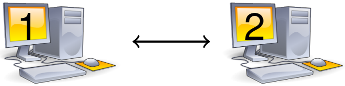
N N

Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p and q are primes)
- Generate p and q using the Miller-Rabin test
- Used in:
  - Encryption schemes
  - Signature schemes
  - Lots of other cryptographic tools
- Paillier Encrypion Scheme

Distributed Generation



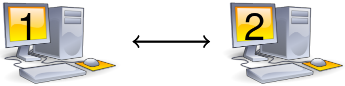
$N = ??$ $N = ??$

Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p and q are primes)
- Generate p and q using the Miller-Rabin test
- Used in:
  - Encryption schemes
  - Signature schemes
  - Lots of other cryptographic tools
- Paillier Encrypion Scheme

Distributed Generation



$N = ??$ $N = ??$

- Introduced by Boneh and Franklin '97
  - 3 Parties (Honest Majority)
  - Passive security
- Other protocols exist.

Introduction: Distributed RSA Key Generation

RSA Composite

- $N = pq$, (p
- Generate p
  Miller-Rabi
- Used in:
  - Encryptic
  - Signature
  - Lots of ot
    tools
- Paillier Encrypion Scheme

- This work
  - Distributed Protocol
  - 2 parties
  - Active security
  - Simulatable Security

Distributed Generation

N =??

oneh and

est Majority)

- Passive security
- Other protocols exist.

Introduction: Threshold Paillier Encryption

Threshold Decryption

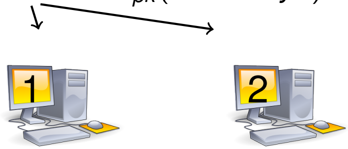
$$c = Enc_{pk}(m = \text{"}hey\text{"})$$

- Many Examples:
  - Threshold RSA
  - Threshold ElGamal
  - etc...

Introduction: Threshold Paillier Encryption

Threshold Decryption

$$c = Enc_{pk}(m = \text{''}hey\text{''})$$



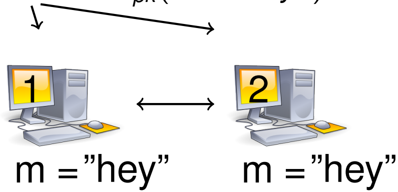
m = ?   m = ?

- Many Examples:
  - Threshold RSA
  - Threshold ElGamal
  - etc...

Introduction: Threshold Paillier Encryption

Threshold Decryption

$c = Enc_{pk}(m = \text{"} hey \text{"})$

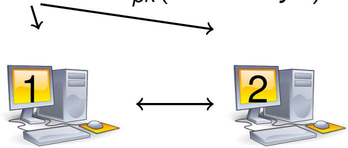


- Many Examples:
  - Threshold RSA
  - Threshold ElGamal
  - etc...

Introduction: Threshold Paillier Encryption

Threshold Decryption

$c = Enc_{pk}(m = \text{"}hey\text{"})$



m = "hey" m = "hey"

- Many Examples:
  - Threshold RSA
  - Threshold ElGamal
  - etc...

Paillier Encryption

- $pk = N$
- $sk = \varphi(N)$
- Additive Homomorphic:

$$Enc_{pk}(m_1 + m_2) = Enc_{pk}(m_1) \cdot Enc_{pk}(m_2)$$

- Useful for MPC/SFE

Introduction: Threshold Paillier Encryption

Threshold Decryption

$c = Enc_{pk}($

m = "hey"

Paillier Encryption

orphic:

$=$

$_{pk}(m_2)$

- This work
  - Threshold Paillier
  - 2 players
  - Active security
  - Simulatable Security

- Many Examples:
  - Threshold RSA
  - Threshold ElGamal
  - etc...
- Useful for MPC/SFE

RSA Composite Generation: Related Work

- Boneh and Franklin ’97
  - Honest majority
  - Pasive security
  - Biprimality test (BF)
- Frankel, Mackenzie, and Yung ’98
  - Honest majority
  - Active security
  - BF biprimality Test
- Poupard and Stern ’98
  - Two party
  - Active Security
  - BF Biprimality Test
  - Not simulatable
- Gilboa ’99
  - Two party
  - Passive Security
  - BF Biprimality Test
- Algesheimer, Camenisch, and Shoup ’02
  - Honest majority
  - Passive Security
  - Miller-Rabin primality test
- Damgård and Mikkelsen ’10
  - Honest majority
  - Actime Security
  - Miller-Rabin like primality test

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute** $N = pq$
4. **Biprimality test**: Are both p and q primes

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute** $N = pq$
4. **Biprimality test**: Are both p and q primes

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute** $N = pq$
4. **Biprimality test**: Are both p and q primes

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute $N = pq$**
4. **Biprimality test**: Are both p and q primes

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute** $N = pq$
4. **Biprimality test**: Are both p and q primes

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute** $N = pq$
4. **Biprimality test**: Are both p and q primes

Trial Division

- Avoid quadratic slowdown:

  One prime at the time: $\frac{1}{\ln(x)}$
  Two primes at the time: $\frac{1}{\ln(x)^2}$

Overview of protocol

1. **Pick random candidates:**
   Pick $p = p_0 + p_1$ and $q = q_0 + q_1$ s.t. $p \equiv q \equiv 3 \pmod 4$.
2. **Trial division:** Distributed trial divide p and q up to a bound B.
   Until p and q succeeds repeat 1 and 2.
3. **Compute** $N = pq$
4. **Biprimality test**: Are both p and q primes

Trial Division

- Avoid quadratic slowdown:

  One prime at the time: $\frac{1}{\ln(x)}$
  Two primes at the time: $\frac{1}{\ln(x)^2}$

Biprimality test

- Faster than distributed primality test, because N is public.

Tools used

- Std. Paillier Encryption (additive homomorphic)
- Additive homomorphic ElGamal
  - $pk = \langle g, h \rangle$, where $g, h \in G_{p'}$
  - $sk = s$ s.t. $h = g^s$
  - $(\alpha, \beta) = Enc_{pk}(m, r) = (g^r, h^r \cdot g^m)$
  - $g^m = Dec_{sk}(\alpha, \beta) = \beta \cdot \alpha^{-s}$
- Threshold additive homomorphic ElGamal
  - $s = s_1 + s_2$
- Integer commitment schemes.
- ZK Proofs

Trial Division

Test if $\alpha | p = p_1 + p_2$

- $c_i = Enc(p_i \bmod \alpha)$, using ElGamal
- Exchange c_i and compute $c = c_1 \cdot c_2$
- If $c = 0$ or $c = \alpha$ then reject p

Speed up

Expected number of Biprimality tests (1024 bit primes):

- ≈ 126000, without trial division
- ≈ 2000, with trial division

Computing $N = pq$

Compute N using Paillier

- P_0: Send $Enc_{pk0}(p_0)$ and $Enc_{pk0}(q_0)$
- P_1: Send

$$Enc_{pk0}(p_0)^{q_1} \cdot Enc_{pk0}(q_0)^{p_1} \cdot Enc_{pk0}(p_1 q_1)$$

$$= Enc_{pk0}((p_0 + p_1)(q_0 + q_1) - (p_0 q_0))$$

- P_0 Compute and send N

Verify computation using ElGamal

Repeat computation using ElGamal and verify that the result is g^N

Biprimality test

The Biprimality test [BF97]

$\gamma^{\frac{\phi(N)}{4}} \equiv \pm 1 \pmod{N}$ for random $\gamma \in \mathbb{Z}_N^*$ and $\mathcal{J}(\gamma) = 1$

Error probability $1/2$

Biprimality test

The Biprimality test [BF97]

$\gamma^{\frac{\phi(N)}{4}} \equiv \pm 1 \pmod{N}$ for random $\gamma \in \mathbb{Z}_N^*$ and $\mathcal{J}(\gamma) = 1$
Error probability $1/2$

The Protocol

1. *Both*: Compute
   $e_0 = Enc_{pk}(\frac{N-(p_0+q_0)+1}{4})$ and
   $e_1 = Enc_{pk}(\frac{-(p_1+q_1)}{4})$ using ElGamal
2. P_0: Send $\gamma_0 = \gamma^{\frac{N-(p_0+q_0)+1}{4}}$
3. P_1: Send $\gamma_1 = \gamma^{\frac{-(p_1+q_1)}{4}}$
4. Both: Prove consistency with e_i
5. Reject N if ($\gamma_0\gamma_1 \bmod N \neq \pm 1$) otherwise repeat ℓ times

Threshold Paillier Scheme - (Updated Version)

Std. Paillier

- $pk = N$, $sk = \varphi(N)$
- $c = Enc_{pk}(m, r) = (1 + N)^m \cdot r^N \bmod N^2$
- $m = Dec_{sk}(c) = \frac{(c^{\phi(N)} \bmod N^2) - 1}{N} \cdot \phi(N)^{-1} \bmod N$

Threshold Paillier Scheme - (Updated Version)

Std. Paillier

- $pk = N$, $sk = \varphi(N)$
- $c = Enc_{pk}(m, r) = (1 + N)^m \cdot r^N \bmod N^2$
- $m = Dec_{sk}(c) = \frac{(c^{\phi(N)} \bmod N^2) - 1}{N} \cdot \phi(N)^{-1} \bmod N$

Threshold version

- d instead of $\phi(N)$, s.t. $d \equiv 1 \pmod N$ and $d \equiv 0 \pmod{\phi(N)}$
- Additive sharing $d = d_0 + d_1$ to compute: $c^d \bmod N^2$

Protocol for Sharing the Private Key $d = d_0 \cdot d_1$

$d \equiv 1 \pmod{N}$ and $d \equiv 0 \pmod{\phi(N)}$

Protocol for Sharing the Private Key $d = d_0 \cdot d_1$

$d \equiv 1 \pmod{N}$ and $d \equiv 0 \pmod{\phi(N)}$

- P_0: Knowledge of $x_0 = N - p_0 - q_0$
- P_1: Knowledge of $x_1 = -p_1 - q_1$

Protocol for Sharing the Private Key $d = d_0 \cdot d_1$

$d \equiv 1 \pmod{N}$ and $d \equiv 0 \pmod{\phi(N)}$

- P_0: Knowledge of $x_0 = N - p_0 - q_0$
- P_1: Knowledge of $x_1 = -p_1 - q_1$
- Similar trick to computing N:
  - P_0 sends P_1 encrypted input
  - P_1 computes and returns result (in this case $d_1 = d + \mathit{blinding}$)
  - To verify ZK-proofs and ElGamal encryptions are used.

Protocol for Decryption $m = Dec(c)$

- P_0: Sends $c_0 = c^{d_0} \bmod N^2$ to P_1
- P_1: Sends $c_1 = c^{d_1} \bmod N^2$ to P_0
- Both: Prove consistency with ElGamal encryption of d_0 and d_1
- Both: Compute:

$$m = ((c_0 \cdot c_1) \bmod N^2 - 1)/N \bmod N$$

Thank You

Please see:

`http://eprint.iacr.org/2011/494`

ALEXANDRA INSTITUTE