Generic Construction of UC-Secure Oblivious Transfer

O. Blazy, C.Chevalier

1. Global Framework
2. Cryptographic Tools
3. 1-out-of-t Oblivious Transfer
4. Instantiation
5. Conclusion

1 Global Framework

2 Cryptographic Tools

3 1-out-of-t Oblivious Transfer

4 Instantiation

5 Conclusion

1 Global Framework

2 Cryptographic Tools

3 1-out-of-t Oblivious Transfer

4 Instantiation

5 Conclusion

1. Global Framework
2. Cryptographic Tools
3. 1-out-of-t Oblivious Transfer
4. Instantiation
5. Conclusion

1. Global Framework
2. Cryptographic Tools
3. 1-out-of-t Oblivious Transfer
4. Instantiation
5. Conclusion

1 Global Framework
- Motivation

2 Cryptographic Tools

3 1-out-of-t Oblivious Transfer

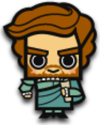
4 Instantiation

5 Conclusion

Conditional Actions

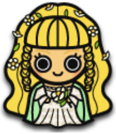
Oblivious Transfer

Database | User



$$\xleftarrow{\mathcal{C}(\text{line})}$$

$$\xrightarrow{\mathcal{DB}[\text{line}]}$$



$\rightsquigarrow$ The User learns the value of line but nothing else.

$\rightsquigarrow$ The Database learns nothing.

Semantic security

- Only the requested line should be learned by the User

Semantic security

- Only the requested line should be learned by the User

Oblivious

- The authority should not learn which line was requested

1 Global Framework

2 Cryptographic Tools
- Encryption Scheme
- Chameleon Hash Scheme
- Smooth Projective Hash Function

3 1-out-of-t Oblivious Transfer

4 Instantiation

5 Conclusion

Definition (Encryption Scheme)

$\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(\mathfrak{K})$: param;
- $\mathsf{KeyGen}(\mathsf{param})$: public *encryption* key pk, private *decryption* key dk;
- $\mathsf{Encrypt}(\mathsf{pk}, m; r)$: ciphertext c on $m \in \mathcal{M}$ and pk;
- $\mathsf{Decrypt}(\mathsf{dk}, c)$: decrypts c under dk.

Indistinguishability under Chosen Ciphertext Attack

Definition (Chameleon Hash Scheme)

$\mathsf{CH} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{CH}, \mathsf{Coll})$:

- $\mathsf{Setup}(\mathfrak{K})$: param;
- $\mathsf{KeyGen}(\mathsf{param})$: outputs the chameleon hash key ck and the trapdoor tk;
- $\mathsf{CH}(\mathsf{ck}, m; r)$: Picks r, and outputs the hash a;
- $\mathsf{Coll}(\mathsf{ck}, m, r, m', \mathsf{tk})$: Takes tk, (m, r) and m', and outputs r' such that $\mathsf{CH}(\mathsf{ck}, m; r) = \mathsf{CH}(\mathsf{ck}, m'; r')$.

Extra Procedures (Verification)

- $\mathsf{VKeyGen}(\mathsf{ck})$: Outputs vk and vtk. $\perp$ or public if publicly verifiable.
- $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, m, a, d, \mathsf{vtk})$: Allows to check that d opens a to m.

Collision Resistance *

Definition (Chameleon Hash Scheme)

$\mathsf{CH} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{CH}, \mathsf{Coll})$:

- $\mathsf{Setup}(\mathfrak{K})$: param;
- $\mathsf{KeyGen}(\mathsf{param})$: outputs the chameleon hash key ck and the trapdoor tk;
- $\mathsf{CH}(\mathsf{ck}, m; r)$: Picks r, and outputs the hash a and verification value d;
- $\mathsf{Coll}(\mathsf{ck}, m, r, m', \mathsf{tk})$: Takes tk, (m, r) and m', and outputs r' such that $\mathsf{CH}(\mathsf{ck}, m; r) = \mathsf{CH}(\mathsf{ck}, m'; r')$.

Extra Procedures (Verification)

- $\mathsf{VKeyGen}(\mathsf{ck})$: Outputs vk and vtk. $\perp$ or public if publicly verifiable.
- $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, m, a, d, \mathsf{vtk})$: Allows to check that d opens a to m.

Collision Resistance *

Definition (Smooth Projective Hash Functions) [CS02]

Let $\{H\}$ be a family of functions:

- X, domain of these functions
- L, subset (a language) of this domain

such that, for any point x in L, $H(x)$ can be computed by using

- either a *secret* hashing key hk: $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$;
- or a *public* projected key hp: $H'(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$

Public mapping $\mathsf{hk} \mapsto \mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$ $\quad$ w witness that $x \in L$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$ $\quad w$ witness that $x \in L$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$ $\quad$ w witness that $x \in L$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

1 Global Framework

2 Cryptographic Tools

3 1-out-of-t Oblivious Transfer
- Definition
- Our Generic Construction
- Security

4 Instantiation

5 Conclusion

Oblivious Transfer [Rab81]

A user U wants to access a line ℓ in a database D composed of t of them:

- U learns nothing more than the value of the line ℓ
- D does not learn which line was accessed by U

Correctness: if U request a single line, he learns it

Security Notions

- Oblivious: D does not know learn which line was accessed ;
- Semantic Security: U does not learn any information about the other lines.

Oblivious Transfer [Rab81]

A user U wants to access a line ℓ in a database D composed of t of them:

- U learns nothing more than the value of the line ℓ
- D does not learn which line was accessed by U

Correctness: if U request a single line, he learns it

Security Notions

- Oblivious: D does not know learn which line was accessed ;
- Semantic Security: U does not learn any information about the other lines.

Oblivious Transfer [Rab81]

A user U wants to access a line ℓ in a database D composed of t of them:

- U learns nothing more than the value of the line ℓ
- D does not learn which line was accessed by U

Correctness: if U request a single line, he learns it

Security Notions

- Oblivious: D does not know learn which line was accessed ;
- Semantic Security: U does not learn any information about the other lines.

Generic bit UC Commitment

- User picks a bit b, random $r, d_{1-b}, \vec{s}$, and computes $(a, d_b) = \mathrm{CH}(\mathsf{ck}, b; r)$
- He then computes $\mathcal{C} = \mathrm{Encrypt}(d_0, d_1; \vec{s})$.

SPHF Compatibility

If the encryption is SPHF friendly, then one can build an SPHF on the language of valid encryption of a chameleon information.
$\mathcal{L}_b = \{c | \exists d_{1-b}, s, \mathrm{Valid}(\mathsf{ck}, \mathsf{vk}, b, a, d_b, \mathsf{vtk}) \wedge c = \mathrm{Encrypt}(d_0, d_1; s)\}$

Generic bit UC Commitment

- User picks a bit b, random $r, d_{1-b}, \vec{s}$, and computes $(a, d_b) = \mathrm{CH}(\mathsf{ck}, b; r)$
- He then computes $\mathcal{C} = \mathrm{Encrypt}(d_0, d_1; \vec{s})$.

SPHF Compatibility

If the encryption is SPHF friendly, then one can build an SPHF on the language of valid encryption of a chameleon information.
$\mathcal{L}_b = \{c | \exists d_{1-b}, s, \mathrm{Valid}(\mathsf{ck}, \mathsf{vk}, b, a, d_b, \mathsf{vtk}) \wedge c = \mathrm{Encrypt}(d_0, d_1; s)\}$

Generic bit UC Commitment

- User picks a bit b, random $r, d_{1-b}, \vec{s}$, and computes $(a, d_b) = \mathrm{CH}(\mathrm{ck}, b; r)$
- He then computes $\mathcal{C} = \mathrm{Encrypt}(d_0, d_1; \vec{s})$.

SPHF Compatibility

If the encryption is SPHF friendly, then one can build an SPHF on the language of valid encryption of a chameleon information.
$\mathcal{L}_b = \{c | \exists d_{1-b}, s, \mathrm{Valid}(\mathrm{ck}, \mathrm{vk}, b, a, d_b, \mathrm{vtk}) \wedge c = \mathrm{Encrypt}(d_0, d_1; s)\}$

Generic bit UC Commitment

- User picks a bit b, random $r, d_{1-b}, \vec{s}$, and computes $(a, d_b) = \mathsf{CH}(\mathsf{ck}, b; r)$
- He then computes $\mathcal{C} = \mathsf{Encrypt}(d_0, d_1; \vec{s})$.

SPHF Compatibility

If the encryption is SPHF friendly, then one can build an SPHF on the language of valid encryption of a chameleon information.
$\mathcal{L}_b = \{c | \exists d_{1-b}, s, \mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, b, a, d_b, \mathsf{vtk}) \wedge c = \mathsf{Encrypt}(d_0, d_1; s)\}$

Generic 1-out-of-t Oblivious Transfer

- User U picks ℓ:
  For each bit, picks random $r_i, d_{1-\ell_i,i}$, and computes $(a_i, d_{\ell_i,i}) = \mathsf{CH}(\mathsf{ck}, \ell_i; r_i)$
  He then computes $\mathcal{C} = \mathsf{Encrypt}(\vec{d}; \vec{s})$ and sends $\mathcal{C}, \vec{a}$.
- For each line L_j, server S computes $\mathsf{hk}_j, \mathsf{hp}_j$, and $H_j = \mathsf{Hash}_{\mathcal{L}_j}(\mathsf{hk}_j, \mathcal{C})$,
  $M_j = H_j \oplus L_j$ and sends M_j, hp_j.
- For the line ℓ, user computes $H'_\ell = \mathsf{ProjHash}_{\mathcal{L}_\ell}(\mathsf{hp}_\ell, \mathcal{C}, \vec{s}_\ell)$, and then
  $L_\ell = M_\ell \oplus H'_\ell$

Generic 1-out-of-t Oblivious Transfer

- User U picks ℓ:
  For each bit, picks random $r_i, d_{1-\ell_i,i}$, and computes $(a_i, d_{\ell_i,i}) = \mathsf{CH}(\mathsf{ck}, \ell_i; r_i)$
  He then computes $\mathcal{C} = \mathsf{Encrypt}(\vec{d}; \vec{s})$ and sends $\mathcal{C}, \vec{a}$.
- For each line L_j, server S computes $\mathsf{hk}_j, \mathsf{hp}_j$, and $H_j = \mathsf{Hash}_{\mathcal{L}_j}(\mathsf{hk}_j, \mathcal{C})$,
  $M_j = H_j \oplus L_j$ and sends M_j, hp_j.
- For the line ℓ, user computes $H'_\ell = \mathsf{ProjHash}_{\mathcal{L}_\ell}(\mathsf{hp}_\ell, \mathcal{C}, \vec{s}_\ell)$, and then
  $L_\ell = M_\ell \oplus H'_\ell$

Generic 1-out-of-t Oblivious Transfer

- User U picks ℓ:
  For each bit, picks random $r_i, d_{1-\ell_i,i}$, and computes $(a_i, d_{\ell_i,i}) = \mathrm{CH}(\mathsf{ck}, \ell_i; r_i)$
  He then computes $\mathcal{C} = \mathrm{Encrypt}(\vec{d}; \vec{s})$ and sends $\mathcal{C}, \vec{a}$.
- For each line L_j, server S computes $\mathsf{hk}_j, \mathsf{hp}_j$, and $H_j = \mathrm{Hash}_{\mathcal{L}_j}(\mathsf{hk}_j, \mathcal{C})$, $M_j = H_j \oplus L_j$ and sends M_j, hp_j.
- For the line ℓ, user computes $H'_\ell = \mathrm{ProjHash}_{\mathcal{L}_\ell}(\mathsf{hp}_\ell, \mathcal{C}, \vec{s}_\ell)$, and then $L_\ell = M_\ell \oplus H'_\ell$

Security Properties

- ✓ Oblivious: IND-CCA security of the encryption scheme;
- ✓ Semantic Security: Smoothness of the SPHF / Collision Resistance of the Chameleon Hash
- ✓ UC simulation: Collision algorithm (Equivocation) of the Chameleon hash

Need an artificial extra-round to handle adaptive corruption
Adds an extra encryption key for a CPA encryption scheme

Security Properties

- ✓ Oblivious: IND-CCA security of the encryption scheme;
- ✓ Semantic Security: Smoothness of the SPHF / Collision Resistance of the Chameleon Hash
- ✓ UC simulation: Collision algorithm (Equivocation) of the Chameleon hash

Need an artificial extra-round to handle adaptive corruption
Adds an extra encryption key for a CPA encryption scheme

1 Global Framework

2 Cryptographic Tools

3 1-out-of-t Oblivious Transfer

4 Instantiation

5 Conclusion

Chameleon Hash: Discrete Logarithm [Ped91]

- KeyGen($\mathfrak{K}$): Outputs $\mathsf{ck} = (g, h)$ $\mathsf{tk} = \alpha = \log_g(h)$;
- VKeyGen(ck): Generates $\mathsf{vk} = f$ and $\mathsf{vtk} = \log_g(f)$
- CH($\mathsf{ck}, \mathsf{vk}, m; r$): $s \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, and outputs $a = h^s g^m$, $d = f^s$.
- Coll(m, s, m', tk): Outputs $s' = s + (m - m')/\alpha$.
- Valid($\mathsf{ck}, \mathsf{vk}, m, a, d, \mathsf{vtk}$): Checks $a \stackrel{?}{=} h^m \cdot d^{1/\mathsf{vtk}}$.

Chameleon Hash: SIS [CHKP10,MP12]

- KeyGen($\mathfrak{K}$): $\vec{A}_0 \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{\mathfrak{K} \times \ell}$, $(\vec{A}_1, \vec{R}_1) \leftarrow \mathsf{GenTrap}^{\mathcal{D}}(1^{\mathfrak{K}}, 1^m, q)$.
  Defines $\mathsf{ck} = (\vec{A}_0, \vec{A}_1)$ and $\mathsf{tk} = \vec{R}_1$.
- VKeyGen(ck): Outputs $\mathsf{vk} = \perp$, $\mathsf{vtk} = \perp$
- CH($\mathsf{ck}, \mathsf{vk}, \vec{M}; \vec{r}$): $\vec{r} \leftarrow D_{\mathbb{Z}^m, s \cdot \omega(\sqrt{\log \mathfrak{K}})}$, $\vec{C} = \vec{A}_0 \vec{M} + \vec{A}_1 \vec{r}$. Returns $\vec{C}, \vec{r}$.
- Coll($\mathsf{tk}, (\vec{M}_0, \vec{r}_0), \vec{M}_1$): Outputs
  $\vec{r}_1 \leftarrow \mathsf{SampleD}(\vec{R}_1, \vec{A}_1, (\vec{A}_0 \vec{M}_0 + \vec{A}_1 \vec{r}_0) - \vec{A}_0 \vec{M}_1), s)$.
- Verif($\mathsf{ck}, \mathsf{vtk}, \vec{M}, \vec{C}, \vec{r}$): $\|\vec{r}\|$ small, and $\vec{C} \stackrel{?}{=} \vec{A}_0 \vec{M} + \vec{A}_1 \vec{r}$.

Chameleon Hash: Discrete Logarithm [Ped91]

- KeyGen($\mathfrak{K}$): Outputs ck $= (g, h)$ tk $= \alpha = \log_g(h)$;
- VKeyGen(ck): Generates vk $= f$ and vtk $= \log_g(f)$
- CH(ck, vk, $m; r$): $s \xleftarrow{\$} \mathbb{Z}_p$, and outputs $a = h^s g^m$, $d = f^s$.
- Coll(m, s, m', tk): Outputs $s' = s + (m - m')/\alpha$.
- Valid(ck, vk, m, a, d, vtk): Checks $a \stackrel{?}{=} h^m \cdot d^{1/\mathsf{vtk}}$.

Chameleon Hash: SIS [CHKP10,MP12]

- KeyGen($\mathfrak{K}$): $\vec{A}_0 \xleftarrow{\$} \mathbb{Z}_q^{\mathfrak{K}\times\ell}$, $(\vec{A}_1, \vec{R}_1) \leftarrow \mathsf{GenTrap}^{\mathcal{D}}(1^{\mathfrak{K}}, 1^m, q)$.
  Defines ck $= (\vec{A}_0, \vec{A}_1)$ and tk $= \vec{R}_1$.
- VKeyGen(ck): Outputs vk $= \perp$, vtk $= \perp$
- CH(ck, vk, $\vec{M}; \vec{r}$): $\vec{r} \leftarrow D_{\mathbb{Z}^m, s\cdot\omega(\sqrt{\log \mathfrak{K}})}$, $\vec{C} = \vec{A}_0\vec{M} + \vec{A}_1\vec{r}$. Returns $\vec{C}, \vec{r}$.
- Coll(tk, $(\vec{M}_0, \vec{r}_0), \vec{M}_1$): Outputs
  $\vec{r}_1 \leftarrow \mathsf{SampleD}(\vec{R}_1, \vec{A}_1, (\vec{A}_0\vec{M}_0 + \vec{A}_1\vec{r}_0) - \vec{A}_0\vec{M}_1), s)$.
- Verif(ck, vtk, $\vec{M}, \vec{C}, \vec{r}$): $\|\vec{r}\|$ small, and $\vec{C} \stackrel{?}{=} \vec{A}_0\vec{M} + \vec{A}_1\vec{r}$.

CCA-2: Cramer Shoup [CS02]

- KeyGen($\mathfrak{K}$): Given g, $x_1, x_2, y_1, y_2, z \xleftarrow{\$} \mathbb{Z}_p$, set $\mathsf{sk} = (x_1, x_2, y_1, y_2, z)$ and $\mathsf{pk} = (g_1, g_2, c_1 = g_1^{x_1} g_2^{x_2}, c_2 = g_1^{y_1} g_2^{y_2}, h_1 = g_1^z, \mathcal{H})$.
- Encrypt($\mathsf{pk}, d; r$): $\mathcal{C} = (u = g_1^r, v = g_2^r, e = h_1^r \cdot d, w = (c_1 c_2^\theta)^r)$, where $\theta = H(\ell, u, v, e)$.
- Decrypt($\mathsf{dk}, \mathcal{C}$): If $w \stackrel{?}{=} u^{x_1+\theta y_1} v^{x_2+\theta y_2}$, then compute $M = e/u^z$.

SPHF on valid encryption of valid chameleon witness

- ProjKG($\mathcal{C}, b$): Computes the projection keys $\mathsf{hp} = h^\lambda f^\kappa, h_1^\kappa g_1^\mu g_2^\nu (c_1 c_2^\beta)^\theta$.
- Hash($\mathcal{C}, \mathsf{hk}$) $H = (\mathcal{C}/g^{m_i})^\lambda \cdot \vec{b}^{\mathsf{hk}}$.
- ProjHash(C, b, hp): The prover will compute $H' = \mathsf{hp}^s \mathsf{hp}^r$.

CCA-2: Cramer Shoup [CS02]

- KeyGen($\mathfrak{K}$): Given g, $x_1, x_2, y_1, y_2, z \xleftarrow{\$} \mathbb{Z}_p$, set sk $= (x_1, x_2, y_1, y_2, z)$ and pk $= (g_1, g_2, c_1 = g_1^{x_1} g_2^{x_2}, c_2 = g_1^{y_1} g_2^{y_2}, h_1 = g_1^z, \mathcal{H})$.
- Encrypt(pk, $d; r$): $\mathcal{C} = (u = g_1^r, v = g_2^r, e = h_1^r \cdot d, w = (c_1 c_2^\theta)^r)$, where $\theta = H(\ell, u, v, e)$.
- Decrypt(dk, $\mathcal{C}$): If $w \stackrel{?}{=} u^{x_1 + \theta y_1} v^{x_2 + \theta y_2}$, then compute $M = e/u^z$.

SPHF on valid encryption of valid chameleon witness

- ProjKG($\mathcal{C}, b$): Computes the projection keys hp $= h^\lambda f^\kappa, h_1^\kappa g_1^\mu g_2^\nu (c_1 c_2^\beta)^\theta$.
- Hash($\mathcal{C}$, hk) $H = (\mathcal{C}/g^{m_i})^\lambda \cdot \vec{b}^{\mathsf{hk}}$.
- ProjHash(C, b, hp): The prover will compute $H' = \mathsf{hp}^s \mathsf{hp}^r$.

CCA-2 ?

- We need an SPHF compatible encryption.
- Only [KV09] is known, and only for approximate SPHF, and is only CCA-1
- However CCA-1 + S-OTS $\Rightarrow$ CCA-2, and Chameleon Hashes gives S-OTS
- Approximate SPHF, requires repetition for perfect line recovery.

CCA-2 ?

- We need an SPHF compatible encryption.
- Only [KV09] is known, and only for approximate SPHF, and is only CCA-1
- However CCA-1 + S-OTS $\Rightarrow$ CCA-2, and Chameleon Hashes gives S-OTS
- Approximate SPHF, requires repetition for perfect line recovery.

CCA-2 ?

- We need an SPHF compatible encryption.
- Only [KV09] is known, and only for approximate SPHF, and is only CCA-1
- However CCA-1 + S-OTS $\Rightarrow$ CCA-2, and Chameleon Hashes gives S-OTS
- Approximate SPHF, requires repetition for perfect line recovery.

CCA-2 ?

- We need an SPHF compatible encryption.
- Only [KV09] is known, and only for approximate SPHF, and is only CCA-1
- However CCA-1 + S-OTS $\Rightarrow$ CCA-2, and Chameleon Hashes gives S-OTS
- Approximate SPHF, requires repetition for perfect line recovery.

- ✓ Generic Framework for 1-out-k Oblivious Transfer
- ✓ Constructions under classical assumptions (DCR, DDH, LWE) in the standard model
- ✓ Proven in the UC framework with adaptive corruptions
- ✓ As efficient as [ABB+13] but without pairings
- ✓ Constant size CRS (contrarily to [PVW08])

- ✓ Generic Framework for 1-out-k Oblivious Transfer
- ✓ Constructions under classical assumptions (DCR, DDH, LWE) in the standard model
- ✓ Proven in the UC framework with adaptive corruptions
- ✓ As efficient as [ABB+13] but without pairings
- ✓ Constant size CRS (contrarily to [PVW08])

- ✓ Generic Framework for 1-out-k Oblivious Transfer
- ✓ Constructions under classical assumptions (DCR, DDH, LWE) in the standard model
- ✓ Proven in the UC framework with adaptive corruptions
- ✓ As efficient as [ABB+13] but without pairings
- ✓ Constant size CRS (contrarily to [PVW08])

- ✓ Generic Framework for 1-out-k Oblivious Transfer
- ✓ Constructions under classical assumptions (DCR, DDH, LWE) in the standard model
- ✓ Proven in the UC framework with adaptive corruptions
- ✓ As efficient as [ABB+13] but without pairings
- ✓ Constant size CRS (contrarily to [PVW08])

✓ Generic Framework for 1-out-k Oblivious Transfer

✓ Constructions under classical assumptions (DCR, DDH, LWE) in the standard model

✓ Proven in the UC framework with adaptive corruptions

✓ As efficient as [ABB+13] but without pairings

✓ Constant size CRS (contrarily to [PVW08])