Achieving Round-Optimal PAKE Protocol

Zengpeng Li[*]

July 19, 2018

1 SPHF From ElGamal

Definition 1.1 (Decisional Diffie-Hellman, (DDH))**.** *The decision-DDH assumption says that, in a group $(p, \mathbb{G}, g)$, where we are given (g^a, g^b, g^c) for unknown random $a, b \leftarrow \mathbb{Z}_p$, it is hard to decide whether $c = ab \pmod q$ (i.e., a real Diffie-Hellman tuple) or $c \xleftarrow{R} \mathbb{Z}_p$ (i.e., a random Diffie-Hellman tuple).*

1.1 ElGamal Scheme

- $\mathsf{params} \leftarrow \mathsf{ElGamal.Setup}(\lambda, G, q, g)$: Takes the security parameter λ, a cyclic group G with prime order q, i.e., $|q| = \lambda$, and the generator g of group G as input, outputs the parameters $\mathsf{params} := (\lambda, G, q, g)$.
- $(sk, pk) \leftarrow \mathsf{ElGamal.KeyGen}(\mathsf{params})$: Takes the params as input, then samples a random $x \in_R \mathbb{Z}_q^*$. Outputs the the secret key $sk := x$ and the public key $pk := (g, h = g^x)$.
- $c \leftarrow \mathsf{ElGamal.Enc}(pk, \mu)$: In order to encrypt the message μ, the algorithm first samples a random $r \in_R \mathbb{Z}_q$, then computes and outputs the ciphertext
$$c := (c_1, c_2) = (g^r, h^r \cdot \mu).$$
- $\mu \leftarrow \mathsf{ElGamal.Dec}(sk, c)$: In order to decrypt the ciphertext, the algorithm computes and outputs $\mu := \frac{c_2}{c_1^x} = \frac{h^r \cdot \mu}{g^r x}$.

It is well known that the ElGamal scheme is IND-CPA-secure under the decisional Diffie-Hellman assumption over G. Hence we omit the further details.

Below we present a SPHF based on $\mathsf{ElGamal}$ scheme, we call it $\mathsf{EG\text{-}SPHF}$. In more detail:

1. $hk \leftarrow \mathsf{HashKG}(\mathsf{params})$: The algorithm samples a_i from $\mathbb{Z}_q$ randomly for $i = 1, 2$, then outputs the hashing key $hk := k = (a_1, a_2)$.
2. $ph \leftarrow \mathsf{ProjKG}(\mathsf{params})$: outputs the projective hashing key $hk := k = (a_1, a_2)$. We stress that, in this case, the projective hashing key is equals the hashing key, hence the projection depends only on the hashing key $k = (a_1, a_2)$.
3. $\mathsf{Hash}(hk, W := (c, \mu))$: (Smooth hash function) The algorithm computes and outputs
$$\mathsf{Hash}((a_1, a_2), (c_1, c_2), \mu) = c_1^{a_1} \cdot (\frac{c_2}{\mu})^{a_2} = g^{ra_1} \cdot h^{ra_2}.$$

[*]Harbin Engineering University, China. Email: zengpengli@hotmail.com

4. $\mathsf{ProjHash}(ph, W := (c, \mu); w)$(Projection)

$$\mathsf{ProjHash}(ph, w) = c_1^{a_1} \cdot c_2^{a_2}$$

Follow the KV construction.

1. $hk \leftarrow \mathsf{HashKG}(\mathsf{params}, pk = (g, h = g^r))$: The algorithm samples a_i from $\mathbb{Z}_q$ randomly for $i = 1, 2$, then outputs the hashing key $hk := k = (a_1, a_2)$.

2. $ph \leftarrow \mathsf{ProjKG}(\mathsf{params}, hk, pk = (g, h = g^r))$: The algorithm takes the hashing key and public key from ElGamal as input, outputs the projective hashing key $ph := p = (g^{a_1}, h^{a_2})$.

3. $\mathsf{Hash}(hk, W := (c, \mu))$: (Smooth hash function) The algorithm computes and outputs

$$\mathsf{Hash}(k = (a_1, a_2), W = ((c_1, c_2), \mu)) = c_1^{a_1} \cdot (\frac{c_2}{\mu})^{a_2} = g^{ra_1} \cdot h^{ra_2}.$$

4. $\mathsf{ProjHash}(ph, W := (c, \mu); w)$(Projection)

$$\mathsf{ProjHash}(p = (g^{a_1}, h^{a_2}), w := r) = (g^{a_1})^r \cdot (h^{a_2})^r.$$

Claim 1.2. *The* EG-SPHF *is a smooth projective hash function for the* $\mathsf{ElGamal}$ *scheme.*

Proof. - **Projective (or Correctness).** This follows from the fact that

$$\begin{aligned} \mathsf{ProjHash}(p = (g^{a_1}, h^{a_2}), w := r) &= (g^{a_1})^r \cdot (h^{a_2})^r \\ &= g^{ra_1} \cdot h^{ra_2} = \mathsf{Hash}(k = (a_1, a_2), W = ((c_1, c_2), \mu)). \end{aligned}$$

- **Smoothness.** Below we prove the smooth property of SPHF. Consider the word $W := \big(c = (c_1, c_2), \mu\big) \notin L$, that mains $c = (c_1, c_2)$ is not an encryption of μ, under the public key pk. Hence the above implies that $(c_1, c_2) = (g^r, h^{r'} \cdot \mu)$ with the witness $r \neq r'$. Next we consider the distribution $\mathsf{Hash}(hk, W) = c_1^{a_1} (\frac{c_2}{\mu})^{a_2} = g^{a_1 r + a_2 x r'}$ given $\mathsf{ProjKG}(hk, pk) = g^{a_1 + a_2 x}$. Since $r \neq r'$, we have that the two equations

$$\begin{aligned} a_1 + a_2 x &= \log_g \mathsf{ProjKG}(hk, pk) \\ a_1 r + a_2 x r' &= \log_g \mathsf{Hash}(hk, W) \end{aligned}$$

are linearly independent. That is, for every choice of $\mathsf{ProjKG}(hk, pk) = g^{a_1 + a_2 x}$ and $\mathsf{Hash}(hk, W)$, there exists a pair (a_1, a_2) that fulfills these equations. Therefore, $\mathsf{ProjKG}(hk, pk)$ provides no information on $\mathsf{Hash}(hk, W)$ and $\mathsf{Hash}(hk, W)$ is uniformly distributed over G, given $\mathsf{ProjKG}(hk, pk)$.

Hence, we conclude that the projective hash function is smooth.

$\square$

2 Labeled Cramer-Shoup Encryption.

Below, we present the Cramer-Shoup Encryption scheme which works in a group $\mathbb{G}$ of prime order p, with two independent generators g and h.

- $(pk, sk) \leftarrow \mathsf{CS.KeyGen}(\mathbb{G}, p, \mathbb{Z}_p)$:

1. Takes a group $\mathbb{G}$ of prime order p as input, then generates two independent generators g and h. Meanwhile, samples five random scalars $x_1, x_2, y_1, y_2, z \leftarrow \mathbb{Z}_q$. Moreover, sample a random collision-resistant hash function H from $\mathcal{H}$.
2. Outputs the secret key $sk = (x_1, x_2, y_1, y_2, z)$ and the public key $pk = (g, h, c = g^{x_1}h^{x_2}, d = g^{y_1}h^{y_2}, f = g^z, H)$.

- $\mathbf{c} \leftarrow \mathsf{CS.Enc}(pk, \mu)$:

1. In order to encrypt the message $\mu \in \mathbb{G}$, the algorithm first generates a label ℓ, then samples a random $r \leftarrow \mathbb{Z}_p$.
2. Computes $u = g^r$, $v = h^r$, $e = f^r \cdot \mu$, and we can hash these values and obtain $\Theta = H(\ell, u, v, e)$. Then computes $\phi = (cd^\Theta)^r$. We note that $(cd^\Theta)^r = \big((g^{x_1}h^{x_2})(g^{y_1}h^{y_2})^\Theta\big)^r = (g^{x_1+\Theta y_1}h^{x_2+\Theta y_2})^r = u^{x_1+\Theta y_1} \cdot v^{x_2+\Theta y_2}$.
3. Outputs the ciphertext $\mathbf{c} = (u, v, e, \phi)$.

- $\mu := \mathsf{CS.Dec}(sk, \mathbf{c})$:

1. **Validity test.** Parses the ciphertext $\mathbf{c}$ into μ, v, e, ϕ, then checks whether the validity of the ciphertext equals $\phi \stackrel{?}{=} \mu^{x_1+\Theta y_1} \cdot v^{x_2+\Theta y_2}$, if yes, then processes the next step.
2. Computes and outputs $\mu := e/\mu^z$.

The correctness is clear, hence we focus on the security analysis.

Theorem 2.1. *If the* DDH *assumption is hard, then the Cramer-Shoup encryption is IND-CCA-secure.*

Proof. We now consider an IND-CCA adversary on single bit.

1. $\mathsf{Hybrid\ H}.0$. This hybrid game is the real IND-CCA game. In this hybrid, we set q_d is the number of decryption queries.

2. $\mathsf{Hybrid\ H}.1$. This hybrid is identical to $\mathsf{Hybrid\ H}.0$ except that we use $f = g^{z_1}h^{z_2}$ to replace $f = g^z$, where z satisfies that $z = z_1 + sz_2$ and $h = g^s$. We stress that, here h is no longer the independent generator. Then, the decryption algorithm works as follows: Once checked the validity of the ciphertext $\phi = \mu^{x_1+\Theta y_1}v^{x_2+\Theta y_2}$, relatively to the label ℓ, then computes $\mu = \frac{e}{(u^{z_1}v^{z_2})}$ (Note that, we should compute $\mu = \frac{e}{u^z}$).

   Remark 2.2. *We remark that,* $u^{z_1}v^{z_2} = (g^r)^{z_1}(h^r)^{z_2} = g^{rz_1}(g^s)^{rz_2} = (g^r)^{z_1+sz_2} = u^z$. *Hence the decryption algorithm outputs the same results for correct ciphertexts.*

   In this setting, any PPT (or unbounded) adversary cannot generate an incorrect ciphertext which can pass the validity test (i.e., $\phi \stackrel{?}{=} \mu^{x_1+\Theta y_1}v^{x_2+\Theta y_2}$) with non-negligible probability. For example, if there exists an incorrect ciphertext, then we set $\phi \stackrel{?}{=} \mu^{x_1+\Theta y_1}v^{x_2+\Theta y_2}$) for $\mu = g^r$ and $v = h^{r'}$ with different random r and $r' \neq r$. For convenience, we set $\delta = r' - r$, then we can obtain $c = g^{x_1}h^{x_2} = g^{x_1+sx_2}$, $d = g^{y_1}h^{y_2} = g^{y_1+sy_2}$, and $\phi = (cd^\Theta)^r \stackrel{?}{=} (g^r)^{x_1+\Theta y_1} \cdot (h^{r'})^{x_2+\Theta y_2} = g^{r(x_1+\Theta y_1)} \cdot g^{sr'(x_2+\Theta y_2)}$. In this setting, we take the discrete logarithm in base g and obtain the following results: $\log c = x_1 + sx_2$, $\log d = y_1 + sy_2$, and $\log \phi = r(x_1 + \Theta y_1) + sr'(x_2 + \Theta y_2)$. Here we note that $\delta = r' - r$, thus we obtain $\log \phi = r(x_1 + \Theta y_1) + s(\delta + r)(x_2 + \Theta y_2) = r(x_1 + \Theta y_1) + sr(x_2 + \Theta y_2) + s\delta(x_2 + \Theta y_2) = r(\log c + \log d) + s\delta(x_2 + \Theta y_2)$. Apparently, c and d cannot reveal any information about the entry of secret key x_2 and y_2. Moreover, due to $s\delta(x_2 + \Theta y_2)$ is unpredictable, thus the correct value for ϕ is also unpredictable. In this setting, a valid incorrect ciphertext is existed with probability less than $1/p$.

Therefore, the distance between the two games is bounded by q_d/p for the number of decryption queries q_d.

3. $\mathsf{Hybrid\ H.2}$. This hybrid is the same as $\mathsf{Hybrid\ H.1}$ except that the challenge ciphertext is generate via a new decryption method. More concretely, consider the Diffie-Hellman tuple $(g, h, \bar{u}, \bar{v})$, then we use $\bar{u}$, $\bar{v}$ and $\bar{e} = (g^{z_1}h^{z_2})^r \cdot \mu = \bar{u}^{z_1}\bar{v}^{z_2} \cdot \mu$ to replace $u = g^r$, $v = h^r$ and $e = f^r \cdot \mu$. Next we obtain the hash value of $\bar{\Theta} = H(\bar{\ell}, (\bar{u}, \bar{v}, \bar{e}))$ and compute $\bar{\phi} = (cd^{\Theta})^r = (\bar{u})^{x_1+\Theta y_1} \cdot (\bar{v})^{x_2+\Theta y_2}$. We note that, the original $\phi = (cd^{\Theta})^r = (g^r)^{x_1+\Theta y_1} \cdot (h^r)^{x_2+\Theta y_2}$ was replaced by $\bar{\phi}$. In this setting, the challenge ciphertext is $\bar{\mathbf{c}} = (\bar{u}, \bar{v}, \bar{e}, \bar{\phi})$ which is sampled randomly and independently from $\mathbb{G}^4$. Therefore, we use the DDH assumption to show that the tuple (u, v, e, ϕ) is indistinguishable from $(\bar{u}, \bar{v}, \bar{e}, \bar{\omega})$.

4. (I don't know, TBA more detail will be found in [**?**] SPHF-Friendly, Non-Interactive Commitment)

□

2.1 The Smooth Projective Hash Function Based On Labeled Cramer-Shoup Encryption

[**?**] Universal one-way hash functions and their cryptographic application

1). Follow the GL construction.

1. $hk \leftarrow \mathsf{HashKG}(\mathsf{params}, pk = (g, h, c = g^{x_1}h^{x_2}, d = g^{y_1}h^{y_2}, f = g^z, H))$: The algorithm samples a_i from $\mathbb{Z}_q$ randomly for $i = 1, \cdots, 4$, then outputs the hashing key $hk := k = (a_1, a_2, a_3, a_4)$.

2. $ph \leftarrow \mathsf{ProjKG}(\mathsf{params}, hk, pk = (g, h, c, d, f, H))$: The algorithm takes the hashing key and public key from Cramer-Shoup scheme as input. *After seeing the ciphertext from Cramer-Shoup scheme, (i.e., $u = g^r$, $v = h^r$, $e = f^r \cdot \mu$)*, the algorithms then computes $\Theta = H(\ell, (u, v, e))$ and outputs the projective hashing key $ph := p = (g^{a_1}, h^{a_2}, f^{a_3}, (cd^{\Theta})^{a_4})$.

3. $\mathsf{Hash}(hk, W := (c, \mu))$: (Smooth hash function) The algorithm takes one of the word $W = (c, \mu)$ over language L as input, then computes and outputs

$$\mathsf{Hash}(k = (a_1, a_2, a_3, a_4), W = ((u, v, e, \phi), \mu) = u^{a_1} \cdot v^{a_2} \cdot (\frac{e}{\mu})^{a_3} \cdot \phi^{a_4} = g^{ra_1} \cdot h^{ra_2} \cdot f^{ra_3} \cdot \phi^{ra_4}.$$

   where $\phi = (cd^{\Theta})^r$.

4. $\mathsf{ProjHash}(ph, W := (c, \mu); w)$(Projection) The algorithm takes the witness w as input.

$$\mathsf{ProjHash}(p = (g^{a_1}, h^{a_2}, f^{a_3}, (cd^{\Theta})^{a_4}), w := r) = (g^{a_1})^r \cdot (h^{a_2})^r \cdot (f^{a_3})^r \cdot ((cd^{\Theta})^{a_4})^r.$$

Claim 2.3. *The* $\mathsf{CS\text{-}SPHF}$ *is a smooth projective hash function for the Cramer-Shoup scheme.*

Proof. More detailed will be found in [**?**] Rosario Gennaro, Yehuda Lindell. A Framework for Password-Based Authenticated Key Exchange. EUROCRYPT 2003: 524-543.

- **Projective (or Correctness).** This follows from the fact that

$$\begin{aligned} \mathsf{ProjHash}(p = (g^{a_1}, h^{a_2}, f^{a_3}, (cd^{\Theta})^{a_4}), w := r) &= (g^{a_1})^r \cdot (h^{a_2})^r \cdot (f^{a_3})^r \cdot ((cd^{\Theta})^{a_4})^r \\ &= \mathsf{Hash}(k = (a_1, a_2, a_3, a_4), W = ((\mu, v, e, \phi), \mu)). \end{aligned}$$

- **Smoothness.** Below we prove the smooth property of SPHF. Consider the word $W := \big(c = (\mu, v, e, \phi), \mu\big) \notin L$, that mains $c = (\mu, v, e, \phi)$ is not an encryption of μ, under the public key pk. Hence the above implies that $(\mu, v, e, \phi) = (g^{r_1}, h^{r_2}, f^{r_3} \cdot \mu)$ with the witness $r_1 \neq r_2 \neq r_3$. Next we consider the distribution $\mathsf{Hash}(hk, W) = c_1^{a_1}(\frac{c_2}{\mu})^{a_2} = g^{a_1 r + a_2 x r'}$ given $\mathsf{ProjKG}(hk, pk) = g^{a_1 + a_2 x}$. Since $r_1 \neq r_2 \neq r_3$, we have that the two equations

$$\begin{aligned} a_1 + a_2 \log_g h + a_3 \log_g f + a_4 \log_g cd^{\Theta} &= \log_g \mathsf{ProjKG}(hk := (g^{a_1}, h^{a_2}, f^{a_3}, (cd^{\Theta})^{a_4}), pk) \\ r_1 a_1 + r_2 a_2 \log_g h + r_3 a_3 \log_g f + a_4 \log_g \phi &= \log_g \mathsf{Hash}(hk := (g^{a_1}, h^{a_2}, f^{a_3}, (cd^{\Theta})^{a_4}), W) \end{aligned}$$

are linearly independent.

We consider three cases:

- $\log_g cd^{\Theta} = \log_g \phi = 0$, i.e., $cd^{\Theta} = \phi = 1$.
- $\log_g cd^{\Theta} = 0$ but $\log_g \phi \neq 0$, i.e., $cd^{\Theta} = 1$ but $\phi \neq 1$. This immediately yields the desired linear independence.
- $\log_g cd^{\Theta} \neq 0$ and $\log_g \phi \neq 0$, i.e., $cd^{\Theta} \neq 1$ and $\phi \neq 1$.

That is, for every choice of $\mathsf{ProjKG}(hk, pk) = g^{a_1 + a_2 x}$ and $\mathsf{Hash}(hk, W)$, there exists a pair (a_1, a_2) that fulfills these equations. Therefore, $\mathsf{ProjKG}(hk, pk)$ provides no information on $\mathsf{Hash}(hk, W)$ and $\mathsf{Hash}(hk, W)$ is uniformly distributed over G, given $\mathsf{ProjKG}(hk, pk)$.

Hence, we conclude that the projective hash function is smooth.

□

2). Follow the Katz-Vaikuntanathan construction.

1. $hk \leftarrow \mathsf{HashKG}(\mathsf{params}, pk = (g, h, c = g^{x_1} h^{x_2}, d = g^{y_1} h^{y_2}, f = g^z, H))$: The algorithm samples a_i from $\mathbb{Z}_q$ randomly for $i = 1, \cdots, 4$, then outputs the hashing key $hk := k = (a_1, a_1', a_2, a_3, a_4)$.
2. $ph \leftarrow \mathsf{ProjKG}(\mathsf{params}, hk, pk = (g, h, c, d, f, H))$: The algorithm takes the hashing key and public key from Cramer-Shoup scheme as input. The the algorithm outputs the projective hashing key $ph := (p_1 = (g^{a_1} \cdot h^{a_2} \cdot f^{a_3} \cdot c^{a_4} \mid p_2 = g^{a_1'} d^{a_4}))$.
3. $\mathsf{Hash}(hk, W := (c, \mu))$: (**1).**Computes the hash value by the smooth hash function.) The algorithm takes one of the word $W = (c, \mu)$ over language L as input, where the ciphertext is $(u = g^r,\ v = h^r,\ e = f^r \cdot \mu, \phi = (cd^{\Theta})^r)$ for $\Theta = H(\ell, u, v, e)$. then computes and outputs

$$\begin{aligned} &\mathsf{Hash}(k = (a_1, a_1', a_2, a_3, a_4), W = ((u, v, e, \phi), \mu) \\ &= u^{(a_1 + \Theta a_1')} \cdot v^{a_2} \cdot (\frac{e}{\mu})^{a_3} \cdot \phi^{a_4} \\ &= g^{r(a_1 + \Theta a_1')} \cdot h^{r a_2} \cdot f^{r a_3} \cdot \phi^{r a_4}. \end{aligned}$$

where $\phi = (cd^{\Theta})^r$.

4. $\mathsf{ProjHash}(ph, W := (c, \mu); w)$(**2).** Computes the hash value by projective hash function.) The algorithm takes the witness w as input.

$$\begin{aligned} &\mathsf{ProjHash}(p := (p_1 = (g^{a_1}, h^{a_2}, f^{a_3}, c^{a_4} \mid p_2 = g^{a_1'} d^{a_4})), W := (c, \mu); w := r) \\ &= \big((g^{a_1})^r \cdot (h^{a_2})^r \cdot (f^{a_3})^r\big) \cdot \big((g^{a_1'} d^{a_4})^{\Theta}\big)^r \\ &= g^{r(a_1 + \Theta a_1')} \cdot h^{r a_2} \cdot f^{r a_3} \cdot \phi^{r a_4} \\ &= (p_1 \cdot p_2^{\Theta})^r. \end{aligned}$$

3 One-Round PAKE Based on DDH Assumption

- Group-based
- Pair-based
- Lattice-based

Common Reference String: public key

Client C with a low-entropy password π

1). Invokes $hk_C \leftarrow \mathsf{HashKG}(\mathsf{params})$
Outputs $hk_C = k_C := (a_1, a_1', a_2, a_3, a_4) \leftarrow \mathbb{Z}_p^5$
2). Invokes $ph_C \leftarrow \mathsf{ProjKG}(\mathsf{params}, hk_C, pk) \in \mathbb{G}_1^2$
Outputs $ph_C = p_C := \left(g^{a_1} h^{a_2} f^{a_3} c^{a_4} \mid g^{a_1'} d^{a_4}\right)$
3). Sets label $\ell_C = C \parallel S \parallel ph_C$
4). Invokes $\mathbf{c}_C \leftarrow \mathsf{Enc}(\mathsf{params}, pk, \pi; r \leftarrow \mathbb{Z}_p)$
Outputs $\mathbf{c}_C := (u, v, e, \phi) \in \mathbb{G}_1^4$
for $u = g^r, v = h^r, e = f^r \pi, \phi = (cd^{\Theta_C})^r$,
and $\Theta_C = H(\ell_C, u, v, e)$.
Erases everything except Θ_C

Server S with a low-entropy password π

1). Invokes $hk_S \leftarrow \mathsf{HashKG}(\mathsf{params})$
Outputs $hk_S = k_S := (\bar{a}_1, \bar{a}_1', \bar{a}_2, \bar{a}_3, \bar{a}_4) \leftarrow \mathbb{Z}_p^5$
2). Invokes $ph_S \leftarrow \mathsf{ProjKG}(\mathsf{params}, hk_S, pk) \in \mathbb{G}_1^2$
Outputs $ph_S = p_S := \left(g^{\bar{a}_1} h^{\bar{a}_2} f^{\bar{a}_3} c^{\bar{a}_4} \mid g^{\bar{a}_1'} d^{\bar{a}_4}\right)$
3). Sets label $\ell_S = S \parallel C \parallel ph_S$
4). Invokes $\mathbf{c}_S \leftarrow \mathsf{Enc}(\mathsf{params}, pk, \pi; \bar{r} \leftarrow \mathbb{Z}_p)$
Outputs $\mathbf{c}_S := (\bar{u}, \bar{v}, \bar{e}, \bar{\phi}) \in \mathbb{G}_1^4$
for $\bar{u} = g^{\bar{r}}, \bar{v} = h^{\bar{r}}, \bar{e} = f^{\bar{r}} \pi, \bar{\phi} = (cd^{\Theta_S})^{\bar{r}}$
and $\Theta_S = H(\ell_S, \bar{u}, \bar{v}, \bar{e})$.
Erases everything except Θ_S

$C \xrightarrow{\mathbf{c}_C, ph_C} S$

$C \xleftarrow{\mathbf{c}_S, ph_S} S$

Client C:

5).Parses ph_S into $g^{\bar{a}_1} h^{\bar{a}_2} f^{\bar{a}_3} c^{\bar{a}_4}$ and $g^{\bar{a}_1'} d^{\bar{a}_4}$.
Parses $\mathbf{c}_S$ into $\bar{u}, \bar{v}, \bar{e}$, and $\bar{\phi}$.
6). Sets label $\ell_S = S \parallel C \parallel ph_S$
7). Extracts $\Theta_S = H(\ell_S, \bar{u}, \bar{v}, \bar{e})$
8). Invokes $\mathsf{ProjHash}(ph_S, W_C := (\mathbf{c}_C, \pi); w_C := r)$
Outputs $p_S := \left(g^{\bar{a}_1} h^{\bar{a}_2} f^{\bar{a}_3} c^{\bar{a}_4}\right)^r \cdot \left(g^{\bar{a}_1'} d^{\bar{a}_4}\right)^{\Theta_C r}$
9). Invokes $\mathsf{Hash}(hk_C, W_S := (\mathbf{c}_S, \pi))$
Outputs $h_C := \bar{u}^{(a_1 + \Theta_S a_1')} \cdot \bar{v}^{a_2} \cdot (\frac{\bar{e}}{\pi})^{a_3} \cdot \bar{\phi}^{a_4}$
i.e., $h_C := g^{\bar{r}(a_1 + \Theta_S a_1')} \cdot h^{\bar{r} a_2} \cdot f^{\bar{r} a_3} \cdot \phi^{\bar{r} a_4}$
Outputs $\mathsf{key} := p_S \cdot h_C$
Erases everything except π and key

Server S:

5).Parses ph_C into $g^{a_1} h^{a_2} f^{a_3} c^{a_4}$ and $g^{a_1'} d^{a_4}$.
Parses $\mathbf{c}_C$ into u, v, e, and ϕ.
6). Sets label $\ell_C = C \parallel S \parallel ph_C$
7). Extracts $\Theta_C = H(\ell_C, u, v, e)$
8). Invokes $\mathsf{ProjHash}(ph_C, W_S := (\mathbf{c}_S, \pi); w_S := \bar{r})$
Outputs $p_C := \left(g^{a_1} h^{a_2} f^{a_3} c^{a_4}\right)^{\bar{r}} \cdot \left(g^{a_1'} d^{a_4}\right)^{\Theta_S \bar{r}}$
9). Invokes $\mathsf{Hash}(hk_S, W_C := (\mathbf{c}_C, \pi))$
$h_S := u^{(\bar{a}_1 + \Theta_C \bar{a}_1')} \cdot v^{\bar{a}_2} \cdot (\frac{e}{\pi})^{\bar{a}_3} \cdot \phi^{\bar{a}_4}$
i.e., $h_S := g^{r(\bar{a}_1 + \Theta_C \bar{a}_1')} \cdot h^{r\bar{a}_2} \cdot f^{r\bar{a}_3} \cdot \phi^{r\bar{a}_4}$
Outputs $\mathsf{key} := p_C \cdot h_S$
Erases everything except π and key

Figure 1: One-round Katz-Vaikuntananthan PAKE Protocol Based On Crammer Shoup Scheme

Let $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.i}(\lambda)$ denote the advantage of $\mathcal{A}$ in experiment $\mathsf{Expt}._i$.

Experiment $\mathsf{Expt}.1$. In this experiment, we change the way $\mathsf{Execute}$ queries are answered. Specifically, the ciphertexts $\mathbf{c}_C$ and $\mathbf{c}_S$ send by the players C (i.e., client) and S (i.e., server) are computed as encryption of 0 instead of being computed as encryptions of the correct password π. We remark that the space of legal passwords is $\{1, \cdots, D\}$, and so 0 is never a valid password. The common session key is computed as

$$\begin{aligned} \mathsf{key}_C &:= \mathsf{Hash}(hk_C, W_S := (\mathbf{c}_S, \pi)) \cdot \mathsf{ProjHash}(ph_S, W_C := (\mathbf{c}_C, \pi); w_C := r) \\ &= \mathsf{Hash}(hk_S, W_C := (\mathbf{c}_C, \pi)) \cdot \mathsf{ProjHash}(ph_C, W_S := (\mathbf{c}_S, \pi); w_S := \bar{r}) \;=\; \mathsf{key}_S. \end{aligned}$$

where both values are computed by using the known hash key hk_C, hk_S and the known projective key ph_C, ph_S. Apparently, the following proof is immediate from semantic security of encryption scheme.

Claim 3.1. $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.0}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.1}(\lambda)|$ *is negligible.*

Experiment $\mathsf{Expt}.2$. In this experiment, we continue to modify the way $\mathsf{Execute}$ queries are answered. More concretely, we sample the common session key $\mathsf{key}_C = \mathsf{key}_S$ uniformly from $\mathbb{G}$.

Claim 3.2. $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.1}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.2}(\lambda)|$ *is negligible.*

Proof. This claim follows the *smoothness* property of SPHF. If there exists a single query that can occur to the $\mathsf{Execute}$ oracle (in either $\mathsf{Expt}.1$ or $\mathsf{Expt}.2$), the adversary can obtain the *transcript*

$(ph_C, \mathbf{c}_C, ph_S, \mathbf{c}_S)$ with $\mathbf{c}_C \leftarrow \mathsf{Enc}(pk, \ell_C, 0; r)$ and $\mathbf{c}_S \leftarrow \mathsf{Enc}(pk, \ell_S, 0; \bar{r})$. We note that, in $\mathsf{Expt}.1$, the common session keys are computed by the following equation

$$\begin{aligned} \mathsf{key}_C &:= \mathsf{Hash}(hk_C, W_S := (\mathbf{c}_S, \pi)) \cdot \mathsf{ProjHash}(ph_S, W_C := (\mathbf{c}_C, \pi); w_C := r) \\ &= \mathsf{Hash}(hk_S, W_C := (\mathbf{c}_C, \pi)) \cdot \mathsf{ProjHash}(ph_C, W_S := (\mathbf{c}_S, \pi); w_S := \bar{r}) \;=\; \mathsf{key}_S. \end{aligned}$$

In the view of client C, the word $W_S := (\mathbf{c}_S, \pi)$ in $X \setminus L$ and not in L. Hence, we use the *smoothness* property to show that $\mathsf{Hash}(hk_C, \ell_S, W_S := (\mathbf{c}_S, \pi))$ is statistically close to uniform g in G. Therefore, in experiment $\mathsf{Expt}.1$, the common session key $\mathsf{key}_C = \mathsf{key}_S$ is statistically close to uniform in $\mathbb{G}$, even conditioned on the given transcript. Since the common session key $\mathsf{key}_C = \mathsf{key}_S$ are chosen uniformly in experiment $\mathsf{Expt}.2$. Hence, this claim follows. □

Remark 3.3. *Before presenting the experiment* $\mathsf{Expt}.3$*, we first distinguish between two possible types of* Send *oracle queries.*

- $\mathsf{Send}_0(C, i, S)$. *It denotes a "prompt" query that causes instance* Π_C^i *of the player* C *to initiate the protocol with user* S. **[linote: In response to a Send_0 query, the adversary is given the message sent by C to S. This query also has the effect of setting $\mathsf{pid}_C^i = S$].**

- $\mathsf{Send}_1(C, i, \mathsf{msg})$. *It means that the adversary* $\mathcal{A}$ *sends the message* μ *to the instance* Π_C^i*. In response, a session key* key_U^i *is computed. (Nothing is output in response to this query, but the value of the computed session key affects a subsequence* Reveal *or* Test *query for instance* Π_C^i*.) For a query* $\mathsf{Send}_1(C, i, \mathsf{msg})$ *with* $\mathsf{pid}_C^i = S$*, we say a valid* msg *is previously used if it was output by a previous oracle query* $\mathsf{Send}_0(S, *, C)$*. In any other case, we say a valid* msg *is adversarially generated. (An invalid message is always ignored by the instance that receives it, and so we assume from now on that* $\mathcal{A}$ *does not send such messages.)*

Experiment $\mathsf{Expt}.3$. In this experiment, we first modify the experiment so that when the public parameters pk are generated the simulator stores the associated secret key sk. (This is just a syntactic change.)

We then modify the way queries to the Send_1 oracle are handled. More concretely, in response to the query $\mathsf{Send}_1(C, i, \mathsf{msg})$ where $\mathsf{msg} = (ph_S, \mathbf{c}_S)$, we distinguish the following three cases (in all the following, let $\mathsf{pid}_C^i = S$, let $\ell_S = (S, C, ph_S)$, and let π be the common session key).

We remark that, in this experiment, the message $\mathsf{msg} = (ph_S, \mathbf{c}_S)$ is never used.

1. If msg is adversarially generated, then compute $\pi_S := \mathsf{Dec}(sk, \ell_S, \mathbf{c}_S)$. Then
   (a) If $\pi_S = \pi$, the simulator declares that $\mathcal{A}$ succeeds and terminates the experiment.
   (b) If $\pi_S \neq \pi$, the simulator chooses key_C^i uniformly from $\mathbb{G}$.

2. If msg is previously used, then in particular the simulator knows a value hk_S such that $ph_S = \mathsf{ProjKG}(hk_S)$. The simulator computes

$$\mathsf{key}_C := \mathsf{Hash}(hk_C, W_S := (\mathbf{c}_S, \pi)) \cdot \mathsf{ProjHash}(ph_S, W_C := (\mathbf{c}_C, \pi); w_C := r),$$

   but using ph_S to compute $\mathsf{ProjHash}(ph_S, W_C := (\mathbf{c}_C, \pi); w_C := r)$ (rather than using the randomness used to generate $\mathbf{c}_C$, as done in $\mathsf{Expt}.2$).

Invalid messages are treated as before, and no session key is computed.

Claim 3.4. $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.2}(\lambda) \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.3}(\lambda) + \mathrm{negl}(\lambda)$.

Proof. Consider the three possible cases described above. The change in Case 1(a) can only increase the advantage of $\mathcal{A}$. The change in Case 1(b) introduces a negligible statistical difference. The analysis is as in Claim 2, except that we now specifically use the fact that *smoothness* holds even under adaptive choice of $W := (\ell_S, \mathbf{c}_S, \pi) \notin L$. The change in Case 2 does not affect the computed value key_C^i since $(\ell_C, \mathbf{c}_C, \pi) \in L$.

□

Experiment Expt.4. Once again we change how Send_1 queries are handled. In response to query $\mathsf{Send}_1(C, i, \mathsf{msg})$ where $\mathsf{msg} = (ph_S, \mathbf{c}_S)$ is previously used.

We remark that, in this experiment, the message $\mathsf{msg} = (ph_S, \mathbf{c}_S)$ is used.

Let $\mathsf{pid}_C^i = S$ and proceed as follows:

- If there exists an instance Π_S^j partnered with Π_C^i (i.e., such that sid_S^j, the transcript of the protocol for instance Π_S^j, is equal to sid_C^i), then set $\mathsf{sid}_S^j = \mathsf{sid}_C^i$.
- Otherwise, choose sid_C^i uniformly from $\mathbb{G}$.

Claim 3.5. $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt.3}}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt.4}}(\lambda)|$ *is negligible.*

Proof. The proof relies on the security of CCA scheme. We first let the upper bound on the number of Send queries issued by adversary $\mathcal{A}$ be ℓ. Consider the following simulator $\mathcal{S}$ interacting in the experiment defined in CCA game.

1. Upon receiving the public key pk and projective key $ph_1, \cdots, ph_\ell$, the simulator $\mathcal{S}$ chooses random passwords π for all players C, S, and runs $\mathcal{A}$ on input pk.
2. In response to $\mathsf{Execute}$ queries as in Expt.2, the simulator $\mathcal{S}$ generates a transcript where the (matching) session keys are chosen uniformly at random, and where the cipheretexts $\mathbf{c}_C$, $\mathbf{c}_S$ are encryption of 0.
3. In response to the Send_0 query $\mathsf{Send}_0(C, *, S)$, the simulator $\mathcal{S}$ first sets $\ell_C := (C, S, ph)$ and submits the word (ℓ_C, π) to the encryption oracle, and receives a ciphertext $\mathbf{c}_C$ along with the hash value $h \leftarrow \mathsf{Hash}(?)$. Then the simulator $\mathcal{S}$ forwards the message $\mathsf{msg} = (ph_C, \mathbf{c}_C)$ to adversary $\mathcal{A}$.
4. In response to a query $\mathsf{Send}_1(C, j, \mathsf{msg})$ for $\mathsf{msg} = (ph_S, \mathbf{c}_S)$, the simulator $\mathcal{S}$ proceeds the following steps:
   - If there exists an instance Π_S^k partnered with Π_U^j, then set $\mathsf{key}_C^j := \mathsf{key}_S^k$.
   - Otherwise, let $\mathsf{pid}_C^j = S$ and $\ell_S = (S, C, ph_S)$, and the adversary $\mathcal{A}$ promote the instance Π_C^j initiate the protocol with the party S by sending the query $\mathsf{Send}_0(C, j, S)$.

     TBA say the query $\mathsf{Send}_0(C, j, S)$ (i.e., the Send query that initiated instance Π_C^j) was the Send_0 query made by adversary $\mathcal{A}$, and resulted in the response $\mathsf{msg} = (ph_C, \mathbf{c}_C)$.

   Below we distinguish the following two cases based on $\mathsf{msg} = (ph_S, \mathbf{c}_S)$. In more detail:

   (a) If msg is previously used, then (by definition) it was output by some previous query $\mathsf{Send}_0(C, *, S)$. That's is to say the Send_0 query made by adversary $\mathcal{A}$, and so $\mathsf{msg} = (ph_C, \mathbf{c}_C)$. Then the simulator computes $\mathsf{key}_C^j = \mathsf{Hash}() \cdot \mathsf{ProjHash}()$.

   (b) If msg is adversarially generated, then simulator $\mathcal{S}$ submits $(\ell_S, \mathbf{c}_S)$ to its decryption oracle and receives in return a value π. If $\pi \neq \pi_{C,S}$, then key_C^i is chosen uniformly from $\mathbb{G}$. If $\pi = \pi_{C,S}$ then $\mathcal{S}$ declares that $\mathcal{A}$ succeeds and terminates the experiment.

5. Lastly, at the end of the experiment, $\mathcal{S}$ outputs 1 if and only if $\mathcal{A}$ succeeds.

On the one hand, there exist two cases for considering b.

- If $b = 0$ then the view of $\mathcal{A}$ in the above execution with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ in $\mathsf{Expt}.3$. This is true since when $b = 0$ it holds in step 4(b), above, that $h = \mathsf{Hash}(\ell_S, \mathbf{c}_r, \pi_{C,S})$ and $h = \mathsf{Hash}(\ell_C, \mathbf{c}_i, \pi_{C,S})$, where $ph_i = \mathsf{ProjKG}(hk_i)$, $ph_r = \mathsf{ProjKG}(hk_r)$, and $\mathbf{c}_i, \mathbf{c}_r$ are encryptions of $\pi_{C,S}$.

- If $b = 1$ then the view of adversary $\mathcal{A}$ in the above execution with simulator $\mathcal{S}$ is identical to the view of $\mathcal{A}$ in $\mathsf{Expt}.4$. To see this, recall that when $b = 1$ all the values $\{h\}$ received by $\mathcal{S}$ are chosen uniformly and independently. We need to show that this generates a uniform and independent distribution on all the session keys computed in step 4(b). Consider a particular session key key computed as in the step 4(b). The only other time the value $h_{i,r}$ could be used in the experiment is if $\mathcal{A}$ queries $\mathsf{Send}_1(S, *, (ph, \mathbf{c}))$ to the instance Π_S^* which sent $(ph_S, \mathbf{c}_S)$. But then Π_S^* and Π_C^j are partnered, and so the session key key_S^* will be set equal to key_U^j (as in $\mathsf{Expt}.4$). Since $h_{i,r}$ is random and used only once to compute a session key in step 5(b), we conclude that (when $b = 1$) any session keys computed in that step are independently uniform in $\mathbb{G}$.

The claim follows from Lemma 1.

□

Experiment $\mathsf{Expt}.5$. In this experiment, we change how Send_0 queries are handled. In response to the query $\mathsf{Send}_0(C, i, S)$, we compute ph as usual but let $\mathbf{c}_C$ be an encryption of 0. The following claim is immediate from IND-CCA-secure CCA scheme.

Claim 3.6. *$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.4}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.5}(\lambda)|$ is negligible.*

Proof. In this experiment, the view of adversary $\mathcal{A}$ is independent of any of the user's passwords until it sends an adversarially generated message that corresponds to an encryption of the correct password (at which point $\mathcal{A}$ succeeds.) It therefore holds that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.5}(\lambda) \leq Q(\lambda)/D$. □

Claims 1-5 thus imply that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Expt}.0}(\lambda) \leq Q(\lambda)/D + \mathrm{negl}(\lambda)$, this completes the proof. □