|  |  |
| --- | --- |
| Protocol Name: |  |
| Protocol Reference: | [Reference number or label] |
| Protocol Type: | [Not always relevant] |
| Protocol Description: |  |
| References: | [Give reference in book or paper] |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | [e.g., P1,P2 or P1,…,Pn, or Committer/Receiver etc. |
| Parties’ Inputs: | [Put each party’s input on a separate line] |
| Parties’ Outputs: | [Put each party’s input on a separate line] |

The ***protocol specification*** describes the instructions of each party as part of the *interaction*:

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (Party 1): |  |
| Step 2 (Party 2): |  |
| Step 3 (Party 3): |  |

The party’s specification describes the instructions of the party from its own point of view (and thus WAIT-FOR-MESSAGE is an often-used instruction here):

|  |  |
| --- | --- |
| Party 1’s Specification | |
| Step 1: |  |
| Step 2: |  |
| Step 3: |  |

|  |  |
| --- | --- |
| Party 2’s Specification | |
| Step 1: |  |
| Step 2: |  |
| Step 3: |  |

Please use the following conventions:

1. All math variables are in bold and italics
2. All math symbols are bold (but not italicized)
3. All descriptions should use the same language and style.
4. A preliminary section should appear explaining what a DLOG description is, and other general remarks.
5. Make sure to use the same notation for sampling values etc. as below
6. Make sure to use consistent, formal expressions. For example, when calling a subprotocol, state: “RUN subprotocol SIGMA\_DLOG with input (g,h) as Verifier”.

The code comments must minimally include the specification.

# Conventions

Include conventions that repeat themselves. For example:

1. A DLOG description (G,q,g) is … We stress that all operations are in the group, unless otherwise explicitly stated.

|  |  |
| --- | --- |
| Protocol Name: | Schnorr’s Protocol for DLOG |
| Protocol Reference: | SIGMA\_DLOG |
| Protocol Type: | Sigma Protocol |
| Protocol Description: | This protocol is used for a prover to convince a verifier that it knows the discrete log of a given value in a given group |
| References: | Protocol 6.1.1, page 148 of Hazay-Lindell |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: (***G,q,g,h***) where (***G,q,g***) is a DLOG description and a parameter ***t*** such that ***2t* < *q*** * P’s private input: a value ***w*∈ *Zq*** such that ***h*=*gw*** |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (both): | V: IF **NOT**   * ***q*** is prime * ***g*** is of order ***q*** * ***h*** ∈ ***G***   Output REJ  P: SAMPLE a random ***r*∈ *Zq*** and COMPUTE ***a* = *gr***  SEND ***a*** to V |
| Step 2 (V): | SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3 (P): | COMPUTE ***z* = *r* + *ew* mod *q***  SEND ***z*** to V |
| Step 4 (both): | V: IF ***gz* = *ahe* mod *p***  OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | SAMPLE a random ***r* ∈ *Zq*** and COMPUTE ***a* = *gr***  SEND ***a*** to V |
| Step 2: | WAIT for message ***e*** from V |
| Step 3: | COMPUTE ***z* = *r* + *ew* mod *q***  SEND ***z*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: | IF **NOT**   * ***q*** is prime * ***g*** is of order ***q*** * ***h*** ∈ ***G***   Output REJ |
| Step 2: | WAIT for message ***a*** from P  SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3: | WAIT for a message ***z*** from P  IF ***gz* = *ahe* mod *p***  OUTPUT ACC  ELSE  OUTPUT REJ |

|  |  |
| --- | --- |
| Protocol Name: | *Σ* Protocol for Diffie-Hellman Tuples |
| Protocol Reference: | SIGMA\_DH |
| Protocol Type: | Sigma Protocol |
| Protocol Description: | This protocol is used for a prover to convince a verifier that a given tuple is a Diffie-Helman tuple. |
| References: | Protocol 6.2.4, page 152 of Hazay-Lindell |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: (***G,q,g,h,u,v***) where (***G,q,g***) is a DLOG description and a parameter ***t*** such that ***2t* < *q*** * P’s private input: a value ***w*∈ *Zq*** such that ***u*=*gw*** and ***v*=*hw*** |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (both): | V: IF **NOT**   * ***q*** is prime * ***g***,***h*** are not of order ***q*** * ***u,v*** ∈ ***G***   Output REJ  P: SAMPLE a random ***r*∈ *Zq*** and COMPUTE ***a* = *gr*** and ***b* = *hr***  SEND ***(a,b)*** to V |
| Step 2 (V): | SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3 (P): | COMPUTE ***z* = *r* + *ew* mod *q***  SEND ***z*** to V |
| Step 4 (both): | V: IF ***gz* = *aue* mod *p*** and ***hz* = *ave***  OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | SAMPLE a random ***r*∈ *Zq*** and COMPUTE ***a* = *gr*** and ***b* = *hr***  SEND ***(a,b)*** to V |
| Step 2: | WAIT for message ***e*** from V |
| Step 3: | COMPUTE ***z* = *r* + *ew* mod *q***  SEND ***z*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: | IF **NOT**   * ***q*** is prime * ***g*** is of order ***q*** * ***h*** ∈ ***G***   Output REJ |
| Step 2: | WAIT for message ***(a,b)*** from P  SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3: | WAIT for a message ***z*** from P  IF ***gz* = *aue* mod *p*** and ***hz* = *ave***  OUTPUT ACC  ELSE  OUTPUT REJ |

|  |  |
| --- | --- |
| Protocol Name: | *Σ Protocol that committed value is as given - Pedersen* |
| Protocol Reference: | SIGMA\_COMMITTED\_VALUE\_PEDERSEN |
| Protocol Type: | Sigma Protocol |
| Protocol Description: |  |
| References: |  |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: ***(G, q, g, α, c, x)*** where (***G,q,g***) is a DLOG description and a parameter ***t*** such that ***2t* < *q*** , ***α* = *ga*** for some ***a ←* Z*q*** and there exist a value ***r*** such that ***c=gr · αx*** * P’s private input: a value ***r*** such that ***c=gr · αx*.** |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (both): | V: IF **NOT**   * ***q*** is prime * ***g*** is not of order ***q*** * ***α,c*** ∈ ***G***   Output REJ  P: SAMPLE a random ***s*∈ *Zq*** and COMPUTE ***d=gs***  SEND ***d*** to V |
| Step 2 (V): | SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3 (P): | COMPUTE ***u* = *s* + *er*****mod *q***  SEND ***u*** to V |
| Step 4 (both): | V: COMPUTE ***b=c/ αx***  IF ***be=gu/gs* mod *p***  OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | SAMPLE a random ***s*∈ *Zq*** and COMPUTE ***d=gs***  SEND ***d*** to V |
| Step 2: | WAIT for message ***e*** from V |
| Step 3: | COMPUTE ***u* = *s* + *er*****mod *q***  SEND ***u*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: | IF **NOT**   * ***q*** is prime * ***g*** is not of order ***q*** * ***α,c*** ∈ ***G***   Output REJ |
| Step 2: | WAIT for message ***d*** from P  SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3: | WAIT for a message ***u*** from P  COMPUTE ***b=c/ αx***  IF ***be=gu/gs* mod *p***  OUTPUT ACC  ELSE  OUTPUT REJ |

|  |  |
| --- | --- |
| Protocol Name: | Σ Protocol for Pedersen Commitments |
| Protocol Reference: | SIGMA\_FOR\_PEDERSEN |
| Protocol Type: | Sigma Protocol |
| Protocol Description: |  |
| References: |  |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: ***(G, q, g, α, c)*** where (***G,q,g***) is a DLOG description and a parameter ***t*** such that ***2t* < *q*** , ***α* = *ga*** for some ***a ←* Z*q***. * P’s private input: values ***x*** and ***r***such that ***c=gr · αx*** |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (both): | V: IF **NOT**   * ***q*** is prime * ***g*** is not of order ***q*** * ***α,c*** ∈ ***G***   Output REJ  P: SAMPLE random values ***t* ∈ *Zq*** and ***s*∈ *Zq*** and COMPUTE ***d=******αtgs***  SEND ***d*** to V |
| Step 2 (V): | SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3 (P): | COMPUTE ***u* = *t* + *ex*****mod *q***and ***v* = *s* + *er*****mod *q***  SEND ***u,v*** to V |
| Step 4 (both): | V: IF ***αugv=dce***  OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | SAMPLE random values ***t* ∈ *Zq*** and ***s*∈ *Zq*** and COMPUTE ***d=******αtgs***  SEND ***d*** to V |
| Step 2: | WAIT for message ***e*** from V |
| Step 3: | COMPUTE ***u* = *t* + *ex*****mod *q***and ***v* = *s* + *er*****mod *q***  SEND ***u,v*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: | : IF **NOT**   * ***q*** is prime * ***g*** is not of order ***q*** * ***α,c*** ∈ ***G***   Output REJ |
| Step 2: | WAIT for message ***d*** from P  SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3: | WAIT for a message ***u,v*** from P  IF ***αugv=dce***  OUTPUT ACC  ELSE  OUTPUT REJ |

|  |  |
| --- | --- |
| Protocol Name: | Σ protocol for El Gamal Commitments |
| Protocol Reference: | SIGMA\_FOR\_PEDERSEN |
| Protocol Type: | Sigma Protocol |
| Protocol Description: |  |
| References: |  |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: ***(G, q, g, h, c=(c1,c2))*** where (***G,q,g***) is a DLOG description and a parameter ***t*** such that ***2t* < *q*** , ***h* = *ga*** for some ***a ←* Z*q***. * P’s private input: values ***m*** and ***r***such that ***c=(gr,hrm)*** |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (both): | V: IF **NOT**   * *g* is a generator of *G* * ***c*1*,c*2*,h*∈ *G***   Output REJ  P: SAMPLE a random value ***s*∈ *Zq*** and COMPUTE ***d=******gs***  SEND ***d*** to V |
| Step 2 (V): | SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3 (P): | COMPUTE ***u* = *s* + *er*****mod *q***  SEND ***u*** to V |
| Step 4 (both): | V: IF ***c1e= gu/d***  OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | SAMPLE a random value ***s*∈ *Zq*** and COMPUTE ***d=******gs***  SEND ***d*** to V |
| Step 2: | WAIT for message ***e*** from V |
| Step 3: | COMPUTE ***u* = *s* + *er*****mod *q***  SEND ***u*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: | IF **NOT**   * *g* is a generator of *G* * ***c*1*,c*2*,h*∈ *G***   Output REJ |
| Step 2: | WAIT for message ***d*** from P  SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3: | WAIT for a message ***u***from P  IF ***c1e= gu/d***  OUTPUT ACC  ELSE  OUTPUT REJ |

|  |  |
| --- | --- |
| Protocol Name: | AND Protocol for Relations *R0*, *R1* Based on *π0 and π1* |
| Protocol Reference: | AND\_SIGMA |
| Protocol Type: | AND of any number of Sigma protocols |
| Protocol Description: | This protocol is used for a prover to convince a verifier that the AND of two statements are true. |
| References: | page 158 of Hazay-Lindell |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: pair ***(x0, x1)*** * P’s private input: a pair *(w0, w1)* such that ***(x0, w0) ∈ R0*** *and*  ***(x1, w1) ∈ R1***. (it might be that ***R0***= ***R1***) |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (P): | P: SEND ***(a0,a1)*** to V |
| Step 2 (V): | SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3 (P): | SEND ***(z0,z1)*** to V |
| Step 4 (both): | V: IF   * transcript ***(a0, e, z0)*** is accepting in **π0**, on inputs ***x0*** * transcript***(a1, e, z1)*** is accepting in ***π1***, on inputs ***x1***.   OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |
|  |  |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | SEND ***(a0,a1)*** to V |
| Step 2: | WAIT for message ***e*** from V |
| Step 3: | SEND ***(z0,z1)*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: |  |
| Step 2: | WAIT for message ***(a0,a1)*** from P  SAMPLE a random challenge ***e* ∈{0, 1}*t***  SEND ***e*** to P |
| Step 3: | WAIT for a message ***(z0,z1)*** from P  IF   * transcripts ***(a0, e, z0)*** is accepting in **π0**, on inputs ***x0*** * transcript***(a1, e, z1)*** is accepting in ***π1***, on inputs ***x1***.   OUTPUT ACC  ELSE  OUTPUT REJ |

|  |  |
| --- | --- |
| Protocol Name: | OR Protocol for Relation *R* Based on *π* |
| Protocol Reference: | OR\_SIGMA |
| Protocol Type: | OR of two Sigma protocols |
| Protocol Description: | This protocol is used for a prover to convince a verifier that it knows a witness to one of the relations without revealing which one. |
| References: | Protocol 6.4.1, page 159 of Hazay-Lindell |

|  |  |
| --- | --- |
| Protocol Parameters | |
| Parties’ Identities: | Prover (P) and Verifier (V) |
| Parties’ Inputs: | * Common input: pair ***(x0, x1)*** * P’s private input: a value ***w***and a bit *b* such that **(*xb,w*) *∈ R*** |
| Parties’ Outputs: | * P: nothing * V: ACC or REJ |

|  |  |
| --- | --- |
| Protocol Specification | |
| Step 1 (P): | P: COMPUTE the first message *ab* in *π*, using (*xb,w*) as input  SAMPLE a random challenge ***e*1*−b* ∈{0, 1}*t***  RUN simulator ***M***on input ***(x1−b, e1−b)***  The output of ***M*** is computed as follows:   * 1. Choose***z1−b*** at random from group ***G***. (e.g. in DL it is ***z ←R Zp\****)   2. Choose***e1−b ←R {0, 1}t***   3. Calculate ***a1−b*** as a function of***(e*1*−b ,z*1*−b* )**. (e.g. in DL it is   ***a1−b = gzh− e1−b***mod *p*)  SEND ***(a0,a1)*** to V |
| Step 2 (V): | SAMPLE a random challenge ***s* ∈{0, 1}*t***  SEND ***s*** to P |
| Step 3 (P): | SET ***eb = s⊕e1−b***  COMPUTE the answer ***zb***in *π* to challenge ***eb***using **(*xb, ab, eb,w*)** as input  SEND ***(e0, z0, e1, z1)*** to V |
| Step 4 (both): | V: IF   * ***e*0 *⊕ e*1 = *s*** * transcripts ***(a0, e, z0)*** is accepting in **π0**, on inputs ***x0*** * transcript***(a1, e, z1)*** is accepting in ***π1***, on inputs ***x1***.   OUTPUT ACC  ELSE  OUTPUT REJ  P: output nothing |
|  |  |

|  |  |
| --- | --- |
| Prover (P) Specification | |
| Step 1: | COMPUTE the first message *ab* in *π*, using (*xb,w*) as input  SAMPLE a random challenge ***e*1*−b* ∈{0, 1}*t***  RUN simulator ***M***on input ***(x1−b, e1−b)***  The output of ***M*** is computed as follows:   * 1. Choose***z1−b*** at random from group ***G***. (e.g. in DL it is ***z ←R Zp\****)   2. Choose***e1−b ←R {0, 1}t***   3. Calculate ***a1−b*** as a function of***(e*1*−b ,z*1*−b* )**. (e.g. in DL it is   ***a1−b = gzh− e1−b***mod *p*)  SEND ***(a0,a1)*** to V |
| Step 2: | WAIT for message ***s*** from V |
| Step 3: | SET ***eb = s⊕e1−b***  COMPUTE the answer ***zb***in *π* to challenge ***eb***using **(*xb, ab, eb,w*)** as input  SEND ***(e0, z0, e1, z1)*** to V |
| Step 4: | OUTPUT nothing |

|  |  |
| --- | --- |
| Verifier (V) Specification | |
| Step 1: |  |
| Step 2: | WAIT for message ***(a0,a1)*** from P  SAMPLE a random challenge ***s* ∈{0, 1}*t***  SEND ***s*** to P |
| Step 3: | WAIT for a message ***(e0, z0, e1, z1)***from P  V: IF   * ***e*0 *⊕ e*1 = *s*** * transcripts ***(a0, e, z0)*** is accepting in **π0**, on inputs ***x0*** * transcript***(a1, e, z1)*** is accepting in ***π1***, on inputs ***x1***.   OUTPUT ACC  ELSE  OUTPUT REJ |