

# Multi-Party Computation

David Aspinall and David Butler

December 14, 2021

## Abstract

We use CryptHOL [1, 6] to consider Multi-Party Computation (MPC) protocols. MPC was first considered in [8] and recent advances in efficiency and an increased demand mean it is now deployed in the real world. Security is considered using the real/ideal world paradigm. We first define security in the semi-honest security setting where parties are assumed not to deviate from the protocol transcript. In this setting we prove multiple Oblivious Transfer (OT) protocols secure and then show security for the gates of the GMW protocol [3]. We then define malicious security, this is a stronger notion of security where parties are assumed to be fully corrupted by an adversary. In this setting we again consider OT.

## Contents

<b>1</b>	<b>Uniform Sampling</b>	<b>4</b>
<b>2</b>	<b>Semi-Honest Security</b>	<b>8</b>
2.1	Security definitions . . . . .	8
2.1.1	Security for deterministic functionalities . . . . .	8
2.1.2	Security definitions for non deterministic functionalities . . . . .	9
2.1.3	Secret sharing schemes . . . . .	10
2.2	Oblivious Transfer functionalities . . . . .	11
2.3	ETP definitions . . . . .	11
2.4	Oblivious transfer constructed from ETPs . . . . .	13
2.4.1	RSA instantiation . . . . .	18
2.5	Noar Pinkas OT . . . . .	23
2.6	1-out-of-2 OT to 1-out-of-4 OT . . . . .	27
2.7	1-out-of-4 OT to GMW . . . . .	34
2.8	Secure multiplication protocol . . . . .	42
2.9	DHH Extension . . . . .	47
<b>3</b>	<b>Malicious Security</b>	<b>49</b>
3.1	Malicious Security Definitions . . . . .	49
3.2	Malicious OT . . . . .	52

```

theory Cyclic-Group-Ext imports
  CryptHOL.CryptHOL
  HOL-Number-Theory.Cong
begin

context cyclic-group begin

lemma generator-pow-order:  $\mathbf{g} [\wedge] \text{order } G = \mathbf{1}$ 
   $\langle \text{proof} \rangle$ 

lemma pow-generator-mod:  $\mathbf{g} [\wedge] (k \text{ mod } \text{order } G) = \mathbf{g} [\wedge] k$ 
   $\langle \text{proof} \rangle$ 

lemma int-nat-pow:
  assumes  $a \geq 0$ 
  shows  $(\mathbf{g} [\wedge] (\text{int } (a :: \text{nat}))) [\wedge] (b :: \text{int}) = \mathbf{g} [\wedge] (a * b)$ 
   $\langle \text{proof} \rangle$ 

lemma pow-generator-mod-int:  $\mathbf{g} [\wedge] ((k :: \text{int}) \text{ mod } \text{order } G) = \mathbf{g} [\wedge] k$ 
   $\langle \text{proof} \rangle$ 

lemma pow-gen-mod-mult:
  shows  $(\mathbf{g} [\wedge] (a :: \text{nat}) \otimes \mathbf{g} [\wedge] (b :: \text{nat})) [\wedge] ((c :: \text{int}) * \text{int } (d :: \text{nat})) = (\mathbf{g} [\wedge] a \otimes \mathbf{g} [\wedge] b) [\wedge] ((c * \text{int } d) \text{ mod } (\text{order } G))$ 
   $\langle \text{proof} \rangle$ 

lemma pow-generator-eq-iff-cong:
  finite (carrier  $G$ )  $\implies \mathbf{g} [\wedge] x = \mathbf{g} [\wedge] y \iff [x = y] (\text{mod } \text{order } G)$ 
   $\langle \text{proof} \rangle$ 

lemma cyclic-group-commute:
  assumes  $a \in \text{carrier } G \ b \in \text{carrier } G$ 
  shows  $a \otimes b = b \otimes a$ 
  (is ?lhs = ?rhs)
   $\langle \text{proof} \rangle$ 

lemma cyclic-group-assoc:
  assumes  $a \in \text{carrier } G \ b \in \text{carrier } G \ c \in \text{carrier } G$ 
  shows  $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ 
  (is ?lhs = ?rhs)
   $\langle \text{proof} \rangle$ 

lemma l-cancel-inv:
  assumes  $h \in \text{carrier } G$ 
  shows  $(\mathbf{g} [\wedge] (a :: \text{nat}) \otimes \text{inv } (\mathbf{g} [\wedge] a)) \otimes h = h$ 
  (is ?lhs = ?rhs)
   $\langle \text{proof} \rangle$ 

lemma inverse-split:

```

```

assumes  $a \in \text{carrier } G$  and  $b \in \text{carrier } G$ 
shows  $\text{inv } (a \otimes b) = \text{inv } a \otimes \text{inv } b$ 
<proof>

lemma inverse-pow-pow:
assumes  $a \in \text{carrier } G$ 
shows  $\text{inv } (a [\wedge] (r::\text{nat})) = (\text{inv } a) [\wedge] r$ 
<proof>

lemma l-neq-1-exp-neq-0:
assumes  $l \in \text{carrier } G$ 
and  $l \neq 1$ 
and  $l = \mathbf{g} [\wedge] (t::\text{nat})$ 
shows  $t \neq 0$ 
<proof>

lemma order-gt-1-gen-not-1:
assumes  $\text{order } G > 1$ 
shows  $\mathbf{g} \neq 1$ 
<proof>

lemma power-swap:  $((\mathbf{g} [\wedge] (\alpha 0::\text{nat})) [\wedge] (r::\text{nat})) = ((\mathbf{g} [\wedge] r) [\wedge] \alpha 0)$ 
(is ?lhs = ?rhs)
<proof>

end

end
theory Number-Theory-Aux imports
  HOL-Number-Theory.Cong
  HOL-Number-Theory.Residues
begin

lemma bezv-inverse:
assumes  $\text{gcd } (e :: \text{nat}) (N :: \text{nat}) = 1$ 
shows  $[\text{nat } e * \text{nat } ((\text{fst } (\text{bezv } e N)) \text{ mod } N) = 1] (\text{mod } \text{nat } N)$ 
<proof>

lemma inverse:
assumes  $\text{gcd } x (q::\text{nat}) = 1$ 
and  $q > 0$ 
shows  $[x * (\text{fst } (\text{bezv } x q)) = 1] (\text{mod } q)$ 
<proof>

lemma prod-not-prime:
assumes prime  $(x::\text{nat})$ 
and prime  $y$ 
and  $x > 2$ 
and  $y > 2$ 

```

**shows**  $\neg \text{prime } ((x-1)*(y-1))$   
 $\langle \text{proof} \rangle$

**lemma** *ex-inverse*:

**assumes** *coprime*:  $\text{coprime } (e :: \text{nat}) ((P-1)*(Q-1))$   
**and** *prime*  $P$   
**and** *prime*  $Q$   
**and**  $P \neq Q$   
**shows**  $\exists d. [e*d = 1] (\text{mod } (P-1)) \wedge d \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *ex-k1-k2*:

**assumes** *coprime*:  $\text{coprime } (e :: \text{nat}) ((P-1)*(Q-1))$   
**and**  $[e*d = 1] (\text{mod } (P-1))$   
**shows**  $\exists k1\ k2. e*d + k1*(P-1) = 1 + k2*(P-1)$   
 $\langle \text{proof} \rangle$

**lemma** *ex-k-mod*:

**assumes** *coprime*:  $\text{coprime } (e :: \text{nat}) ((P-1)*(Q-1))$   
**and**  $P \neq Q$   
**and** *prime*  $P$   
**and** *prime*  $Q$   
**and**  $d \neq 0$   
**and**  $[e*d = 1] (\text{mod } (P-1))$   
**shows**  $\exists k. e*d = 1 + k*(P-1)$   
 $\langle \text{proof} \rangle$

**lemma** *fermat-little*:

**assumes** *prime*  $(P :: \text{nat})$   
**shows**  $[x^P = x] (\text{mod } P)$   
 $\langle \text{proof} \rangle$

**end**

## 1 Uniform Sampling

Here we prove different one time pad lemmas based on uniform sampling we require throughout our proofs.

**theory** *Uniform-Sampling*

**imports**

*CryptHOL.Cyclic-Group-SPMF*

*HOL-Number-Theory.Cong*

*CryptHOL.List-Bits*

**begin**

If  $q$  is a prime we can sample from the units.

**definition** *sample-uniform-units*  $:: \text{nat} \Rightarrow \text{nat } \text{spmf}$

**where** *sample-uniform-units*  $q = \text{spmf-of-set } (\{..< q\} - \{0\})$

**lemma** *set-spmf-sampl-uni-units* [*simp*]:  $set\text{-}spmf\ (sample\text{-}uniform\text{-}units\ q) = \{.. $q$ \} - \{0\}$   
 ⟨*proof*⟩

**lemma** *lossless-sample-uniform-units*:  
**assumes**  $q > 1$   
**shows**  $lossless\text{-}spmf\ (sample\text{-}uniform\text{-}units\ q)$   
 ⟨*proof*⟩

General lemma for mapping using uniform sampling from units.

**lemma** *one-time-pad-units*:  
**assumes** *inj-on*:  $inj\text{-}on\ f\ (\{.. $q$ \} - \{0\})$   
**and** *sur*:  $f\ ' (\{.. $q$ \} - \{0\}) = (\{.. $q$ \} - \{0\})$   
**shows**  $map\text{-}spmf\ f\ (sample\text{-}uniform\text{-}units\ q) = (sample\text{-}uniform\text{-}units\ q)$   
 (**is**  $?lhs = ?rhs$ )  
 ⟨*proof*⟩

General lemma for mapping using uniform sampling.

**lemma** *one-time-pad*:  
**assumes** *inj-on*:  $inj\text{-}on\ f\ \{.. $q$ \}$   
**and** *sur*:  $f\ ' \{.. $q$ \} = \{.. $q$ \}$   
**shows**  $map\text{-}spmf\ f\ (sample\text{-}uniform\ q) = (sample\text{-}uniform\ q)$   
 (**is**  $?lhs = ?rhs$ )  
 ⟨*proof*⟩

The addition map case.

**lemma** *inj-add*:  
**assumes**  $x: x < q$   
**and**  $x': x' < q$   
**and** *map*:  $((y :: nat) + x) \bmod q = (y + x') \bmod q$   
**shows**  $x = x'$   
 ⟨*proof*⟩

**lemma** *inj-uni-samp-add*:  $inj\text{-}on\ (\lambda(b :: nat). (y + b) \bmod q)\ \{.. $q$ \}$   
 ⟨*proof*⟩

**lemma** *surj-uni-samp*:  
**assumes** *inj*:  $inj\text{-}on\ (\lambda(b :: nat). (y + b) \bmod q)\ \{.. $q$ \}$   
**shows**  $(\lambda(b :: nat). (y + b) \bmod q)\ ' \{.. $q$ \} = \{.. $q$ \}$   
 ⟨*proof*⟩

**lemma** *samp-uni-plus-one-time-pad*:  
**shows**  $map\text{-}spmf\ (\lambda b. (y + b) \bmod q)\ (sample\text{-}uniform\ q) = (sample\text{-}uniform\ q)$   
 ⟨*proof*⟩

The multiplication map case.

**lemma** *inj-mult*:

**assumes** *coprime*:  $\text{coprime } x \ (q::\text{nat})$   
**and**  $y: y < q$   
**and**  $y': y' < q$   
**and**  $\text{map}: x * y \text{ mod } q = x * y' \text{ mod } q$   
**shows**  $y = y'$   
 $\langle \text{proof} \rangle$

**lemma** *inj-on-mult*:  
**assumes** *coprime*:  $\text{coprime } x \ (q::\text{nat})$   
**shows** *inj-on*  $(\lambda b. x*b \text{ mod } q) \ \{..<q\}$   
 $\langle \text{proof} \rangle$

**lemma** *surj-on-mult*:  
**assumes** *coprime*:  $\text{coprime } x \ (q::\text{nat})$   
**and** *inj*: *inj-on*  $(\lambda b. x*b \text{ mod } q) \ \{..<q\}$   
**shows**  $(\lambda b. x*b \text{ mod } q) \ \{..<q\} = \{..<q\}$   
 $\langle \text{proof} \rangle$

**lemma** *mult-one-time-pad*:  
**assumes** *coprime*:  $\text{coprime } x \ q$   
**shows** *map-spmf*  $(\lambda b. x*b \text{ mod } q) \ (\text{sample-uniform } q) = (\text{sample-uniform } q)$   
 $\langle \text{proof} \rangle$

The multiplication map for sampling from units.

**lemma** *inj-on-mult-units*:  
**assumes**  $1: \text{coprime } x \ (q::\text{nat})$  **shows** *inj-on*  $(\lambda b. x*b \text{ mod } q) \ (\{..<q\} - \{0\})$   
 $\langle \text{proof} \rangle$

**lemma** *surj-on-mult-units*:  
**assumes** *coprime*:  $\text{coprime } x \ (q::\text{nat})$   
**and** *inj*: *inj-on*  $(\lambda b. x*b \text{ mod } q) \ (\{..<q\} - \{0\})$   
**shows**  $(\lambda b. x*b \text{ mod } q) \ (\{..<q\} - \{0\}) = (\{..<q\} - \{0\})$   
 $\langle \text{proof} \rangle$

**lemma** *mult-one-time-pad-units*:  
**assumes** *coprime*:  $\text{coprime } x \ q$   
**shows** *map-spmf*  $(\lambda b. x*b \text{ mod } q) \ (\text{sample-uniform-units } q) = \text{sample-uniform-units } q$   
 $\langle \text{proof} \rangle$

Addition and multiplication map.

**lemma** *samp-uni-add-mult*:  
**assumes** *coprime*:  $\text{coprime } x \ (q::\text{nat})$   
**and**  $xa: xa < q$   
**and**  $ya: ya < q$   
**and**  $\text{map}: (y + x * xa) \text{ mod } q = (y + x * ya) \text{ mod } q$   
**shows**  $xa = ya$   
 $\langle \text{proof} \rangle$

**lemma** *inj-on-add-mult*:

**assumes** *coprime*: *coprime* *x* (*q*::*nat*)  
**shows** *inj-on* ( $\lambda b. (y + x*b) \bmod q$ )  $\{..<q\}$   
*<proof>*

**lemma** *surj-on-add-mult*: **assumes** *coprime*: *coprime* *x* (*q*::*nat*) **and** *inj*: *inj-on*  
( $\lambda b. (y + x*b) \bmod q$ )  $\{..<q\}$

**shows** ( $\lambda b. (y + x*b) \bmod q$ ) ‘  $\{..<q\} = \{..<q\}$   
*<proof>*

**lemma** *add-mult-one-time-pad*: **assumes** *coprime*: *coprime* *x* *q*

**shows** *map-spmf* ( $\lambda b. (y + x*b) \bmod q$ ) (*sample-uniform* *q*) = (*sample-uniform*  
*q*)  
*<proof>*

Subtraction Map.

**lemma** *inj-minus*:

**assumes** *x*: (*x* :: *nat*) < *q*  
**and** *ya*: *ya* < *q*  
**and** *map*: ( $y + q - x$ ) *mod* *q* = ( $y + q - ya$ ) *mod* *q*  
**shows** *x* = *ya*  
*<proof>*

**lemma** *inj-on-minus*: *inj-on* ( $\lambda(b :: nat). (y + (q - b)) \bmod q$ )  $\{..<q\}$   
*<proof>*

**lemma** *surj-on-minus*:

**assumes** *inj*: *inj-on* ( $\lambda(b :: nat). (y + (q - b)) \bmod q$ )  $\{..<q\}$   
**shows** ( $\lambda(b :: nat). (y + (q - b)) \bmod q$ ) ‘  $\{..<q\} = \{..<q\}$   
*<proof>*

**lemma** *samp-uni-minus-one-time-pad*:

**shows** *map-spmf*( $\lambda b. (y + (q - b)) \bmod q$ ) (*sample-uniform* *q*) = (*sample-uniform*  
*q*)  
*<proof>*

**lemma** *not-coin-flip*: *map-spmf* ( $\lambda a. \neg a$ ) *coin-spmf* = *coin-spmf*

*<proof>*

**lemma** *xor-uni-samp*: *map-spmf*( $\lambda b. y \oplus b$ ) (*coin-spmf*) = *map-spmf*( $\lambda b. b$ )  
(*coin-spmf*)

(**is** *?lhs* = *?rhs*)  
*<proof>*

**end**

## 2 Semi-Honest Security

We follow the security definitions for the semi honest setting as described in [5]. In the semi honest model the parties are assumed not to deviate from the protocol transcript. Semi honest security guarantees that no information is leaked during the running of the protocol.

### 2.1 Security definitions

```
theory Semi-Honest-Def imports
  CryptHOL.CryptHOL
begin
```

#### 2.1.1 Security for deterministic functionalities

```
locale sim-det-def =
  fixes R1 :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view1 spmf
    and S1 :: 'msg1  $\Rightarrow$  'out1  $\Rightarrow$  'view1 spmf
    and R2 :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view2 spmf
    and S2 :: 'msg2  $\Rightarrow$  'out2  $\Rightarrow$  'view2 spmf
    and funct :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  ('out1  $\times$  'out2) spmf
    and protocol :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  ('out1  $\times$  'out2) spmf
  assumes lossless-R1: lossless-spmf (R1 m1 m2)
    and lossless-S1: lossless-spmf (S1 m1 out1)
    and lossless-R2: lossless-spmf (R2 m1 m2)
    and lossless-S2: lossless-spmf (S2 m2 out2)
    and lossless-funct: lossless-spmf (funct m1 m2)
  begin

  type-synonym 'view' adversary-det = 'view'  $\Rightarrow$  bool spmf

  definition correctness m1 m2  $\equiv$  (protocol m1 m2 = funct m1 m2)

  definition adv-P1 :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view1 adversary-det  $\Rightarrow$  real
    where adv-P1 m1 m2 D  $\equiv$  |(spmf (R1 m1 m2  $\ggg$  D) True)
      - spmf (funct m1 m2  $\ggg$  ( $\lambda$  (o1, o2). S1 m1 o1  $\ggg$  D)) True|

  definition perfect-sec-P1 m1 m2  $\equiv$  (R1 m1 m2 = funct m1 m2  $\ggg$  ( $\lambda$  (s1, s2).
    S1 m1 s1))

  definition adv-P2 :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view2 adversary-det  $\Rightarrow$  real
    where adv-P2 m1 m2 D = |spmf (R2 m1 m2  $\ggg$  ( $\lambda$  view. D view)) True
      - spmf (funct m1 m2  $\ggg$  ( $\lambda$  (o1, o2). S2 m2 o2  $\ggg$  ( $\lambda$  view. D view)))
    True|

  definition perfect-sec-P2 m1 m2  $\equiv$  (R2 m1 m2 = funct m1 m2  $\ggg$  ( $\lambda$  (s1, s2).
    S2 m2 s2))
```

We also define the security games (for Party 1 and 2) used in EasyCrypt to



define semi honest security for Party 1. We then show the two definitions are equivalent.

**definition** *P1-game-alt* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view1 adversary-det  $\Rightarrow$  bool spmf  
**where** *P1-game-alt* m1 m2 D = do {  
 b  $\leftarrow$  coin-spmf;  
 (out1, out2)  $\leftarrow$  funct m1 m2;  
 rview :: 'view1  $\leftarrow$  R1 m1 m2;  
 sview :: 'view1  $\leftarrow$  S1 m1 out1;  
 b'  $\leftarrow$  D (if b then rview else sview);  
 return-spmf (b = b')}

**definition** *adv-P1-game* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view1 adversary-det  $\Rightarrow$  real  
**where** *adv-P1-game* m1 m2 D = |2\*(spmf (P1-game-alt m1 m2 D) True) - 1|

We show the two definitions are equivalent

**lemma** *equiv-defs-P1*:

**assumes** *lossless-D*:  $\forall$  view. *lossless-spmf* ((D:: 'view1 adversary-det) view)  
**shows** *adv-P1-game* m1 m2 D = *adv-P1* m1 m2 D  
**including** *monad-normalisation*  
 <proof>

**definition** *P2-game-alt* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view2 adversary-det  $\Rightarrow$  bool spmf  
**where** *P2-game-alt* m1 m2 D = do {  
 b  $\leftarrow$  coin-spmf;  
 (out1, out2)  $\leftarrow$  funct m1 m2;  
 rview :: 'view2  $\leftarrow$  R2 m1 m2;  
 sview :: 'view2  $\leftarrow$  S2 m2 out2;  
 b'  $\leftarrow$  D (if b then rview else sview);  
 return-spmf (b = b')}

**definition** *adv-P2-game* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'view2 adversary-det  $\Rightarrow$  real  
**where** *adv-P2-game* m1 m2 D = |2\*(spmf (P2-game-alt m1 m2 D) True) - 1|

**lemma** *equiv-defs-P2*:

**assumes** *lossless-D*:  $\forall$  view. *lossless-spmf* ((D:: 'view2 adversary-det) view)  
**shows** *adv-P2-game* m1 m2 D = *adv-P2* m1 m2 D  
**including** *monad-normalisation*  
 <proof>

**end**

## 2.1.2 Security definitions for non deterministic functionalities

**locale** *sim-non-det-def* =

**fixes** *R1* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  ('view1  $\times$  ('out1  $\times$  'out2)) spmf  
**and** *S1* :: 'msg1  $\Rightarrow$  'out1  $\Rightarrow$  'view1 spmf  
**and** *Out1* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  'out1  $\Rightarrow$  ('out1  $\times$  'out2) spmf — takes the  
 input of the other party so can form the outputs of parties  
**and** *R2* :: 'msg1  $\Rightarrow$  'msg2  $\Rightarrow$  ('view2  $\times$  ('out1  $\times$  'out2)) spmf

```

and  $S2 :: 'msg2 \Rightarrow 'out2 \Rightarrow 'view2 \text{ spmf}$ 
and  $Out2 :: 'msg2 \Rightarrow 'msg1 \Rightarrow 'out2 \Rightarrow ('out1 \times 'out2) \text{ spmf}$ 
and  $funct :: 'msg1 \Rightarrow 'msg2 \Rightarrow ('out1 \times 'out2) \text{ spmf}$ 
begin

type-synonym ( $'view', 'out1', 'out2'$ )  $adversary\text{-non-det} = ('view' \times ('out1' \times 'out2')) \Rightarrow \text{bool spmf}$ 

definition  $Ideal1 :: 'msg1 \Rightarrow 'msg2 \Rightarrow 'out1 \Rightarrow ('view1 \times ('out1 \times 'out2)) \text{ spmf}$ 
where  $Ideal1\ m1\ m2\ out1 = \text{do } \{$ 
   $view1 :: 'view1 \leftarrow S1\ m1\ out1;$ 
   $out1 \leftarrow Out1\ m1\ m2\ out1;$ 
   $\text{return-spmf } (view1, out1)\}$ 

definition  $Ideal2 :: 'msg2 \Rightarrow 'msg1 \Rightarrow 'out2 \Rightarrow ('view2 \times ('out1 \times 'out2)) \text{ spmf}$ 
where  $Ideal2\ m2\ m1\ out2 = \text{do } \{$ 
   $view2 :: 'view2 \leftarrow S2\ m2\ out2;$ 
   $out2 \leftarrow Out2\ m2\ m1\ out2;$ 
   $\text{return-spmf } (view2, out2)\}$ 

definition  $adv\text{-}P1 :: 'msg1 \Rightarrow 'msg2 \Rightarrow ('view1, 'out1, 'out2) \text{ adversary-non-det}$ 
 $\Rightarrow \text{real}$ 
where  $adv\text{-}P1\ m1\ m2\ D \equiv |(spm\text{f } (R1\ m1\ m2 \gg (\lambda\ view.\ D\ view))\ True) -$ 
 $spm\text{f } (funct\ m1\ m2 \gg (\lambda\ (o1, o2).\ Ideal1\ m1\ m2\ o1 \gg (\lambda\ view.\ D\ view)))\ True|$ 

definition  $perfect\text{-sec-}P1\ m1\ m2 \equiv (R1\ m1\ m2 = funct\ m1\ m2 \gg (\lambda\ (s1, s2).\$ 
 $Ideal1\ m1\ m2\ s1))$ 

definition  $adv\text{-}P2 :: 'msg1 \Rightarrow 'msg2 \Rightarrow ('view2, 'out1, 'out2) \text{ adversary-non-det}$ 
 $\Rightarrow \text{real}$ 
where  $adv\text{-}P2\ m1\ m2\ D = |spm\text{f } (R2\ m1\ m2 \gg (\lambda\ view.\ D\ view))\ True - spm\text{f}$ 
 $(funct\ m1\ m2 \gg (\lambda\ (o1, o2).\ Ideal2\ m2\ m1\ o2 \gg (\lambda\ view.\ D\ view)))\ True|$ 

definition  $perfect\text{-sec-}P2\ m1\ m2 \equiv (R2\ m1\ m2 = funct\ m1\ m2 \gg (\lambda\ (s1, s2).\$ 
 $Ideal2\ m2\ m1\ s2))$ 

end

2.1.3 Secret sharing schemes

locale  $secret\text{-sharing-scheme} =$ 
fixes  $share :: 'input\text{-out} \Rightarrow ('share \times 'share) \text{ spmf}$ 
and  $reconstruct :: ('share \times 'share) \Rightarrow 'input\text{-out} \text{ spmf}$ 
and  $F :: ('input\text{-out} \Rightarrow 'input\text{-out} \Rightarrow 'input\text{-out} \text{ spmf}) \text{ set}$ 
begin

definition  $sharing\text{-correct}\ input \equiv (share\ input \gg (\lambda\ (s1, s2).\ reconstruct\ (s1, s2)))$ 
 $= \text{return-spmf } input$ 

```

**definition** *correct-share-eval input1 input2*  $\equiv (\forall \text{ gate-eval} \in F.$   
 $\exists \text{ gate-protocol} :: ('share \times 'share) \Rightarrow ('share \times 'share) \Rightarrow ('share \times$   
 $'share) \text{ spmf.}$   
 $\text{share input1} \gg= (\lambda (s1,s2). \text{share input2}$   
 $\gg= (\lambda (s3,s4). \text{gate-protocol } (s1,s3) (s2,s4)$   
 $\gg= (\lambda (S1,S2). \text{reconstruct } (S1,S2)))) = \text{gate-eval input1}$   
 $\text{input2})$

**end**

**end**

## 2.2 Oblivious Transfer functionalities

Here we define the functionalities for 1-out-of-2 and 1-out-of-4 OT.

**theory** *OT-Functionalities* **imports**

*CryptHOL.CryptHOL*

**begin**

**definition** *funct-OT-12*  $:: ('a \times 'a) \Rightarrow \text{bool} \Rightarrow (\text{unit} \times 'a) \text{ spmf}$

**where** *funct-OT-12 input<sub>1</sub>*  $\sigma = \text{return-spmf } ((), \text{ if } \sigma \text{ then } (\text{snd input}_1) \text{ else } (\text{fst input}_1))$

**lemma** *lossless-funct-OT-12: lossless-spmf (funct-OT-12 msgs  $\sigma$ )*

*<proof>*

**definition** *funct-OT-14*  $:: ('a \times 'a \times 'a \times 'a) \Rightarrow (\text{bool} \times \text{bool}) \Rightarrow (\text{unit} \times 'a) \text{ spmf}$

**where** *funct-OT-14 M C*  $= \text{do } \{$   
 $\text{let } (c0,c1) = C;$   
 $\text{let } (m00, m01, m10, m11) = M;$   
 $\text{return-spmf } ((), \text{if } c0 \text{ then } (\text{if } c1 \text{ then } m11 \text{ else } m10) \text{ else } (\text{if } c1 \text{ then } m01 \text{ else } m00))\}$

**lemma** *lossless-funct-14-OT: lossless-spmf (funct-OT-14 M C)*

*<proof>*

**end**

## 2.3 ETP definitions

We define Extended Trapdoor Permutations (ETPs) following [5] and [2]. In particular we consider the property of Hard Core Predicates (HCPs).

**theory** *ETP* **imports**

*CryptHOL.CryptHOL*

**begin**

**type-synonym** *(index,range) dist2*  $= (\text{bool} \times 'index \times \text{bool} \times \text{bool}) \Rightarrow \text{bool} \text{ spmf}$

**type-synonym** ('index,'range) advP2 = 'index  $\Rightarrow$  bool  $\Rightarrow$  bool  $\Rightarrow$  ('index,'range)  
 dist2  $\Rightarrow$  'range  $\Rightarrow$  bool spmf

**locale** etp =

**fixes**  $I :: ('index \times 'trap) \text{ spmf}$  — samples index and trapdoor  
**and**  $\text{domain} :: 'index \Rightarrow 'range \text{ set}$   
**and**  $\text{range} :: 'index \Rightarrow 'range \text{ set}$   
**and**  $F :: 'index \Rightarrow ('range \Rightarrow 'range)$  — permutation  
**and**  $F_{inv} :: 'index \Rightarrow 'trap \Rightarrow 'range \Rightarrow 'range$  — must be efficiently computable  
**and**  $B :: 'index \Rightarrow 'range \Rightarrow \text{bool}$  — hard core predicate  
**assumes**  $\text{dom-eq-ran}: y \in \text{set-spmf } I \longrightarrow \text{domain } (fst\ y) = \text{range } (fst\ y)$   
**and**  $\text{finite-range}: y \in \text{set-spmf } I \longrightarrow \text{finite } (\text{range } (fst\ y))$   
**and**  $\text{non-empty-range}: y \in \text{set-spmf } I \longrightarrow \text{range } (fst\ y) \neq \{\}$   
**and**  $\text{bij-betw}: y \in \text{set-spmf } I \longrightarrow \text{bij-betw } (F\ (fst\ y))\ (\text{domain } (fst\ y))\ (\text{range } (fst\ y))$   
**and**  $\text{lossless-I}: \text{lossless-spmf } I$   
**and**  $F\text{-f-inv}: y \in \text{set-spmf } I \longrightarrow x \in \text{range } (fst\ y) \longrightarrow F_{inv}\ (fst\ y)\ (snd\ y)\ (F\ (fst\ y)\ x) = x$   
**begin**

**definition**  $S :: 'index \Rightarrow 'range \text{ spmf}$   
**where**  $S\ \alpha = \text{spmof-of-set } (\text{range } \alpha)$

**lemma**  $\text{lossless-S}: y \in \text{set-spmf } I \longrightarrow \text{lossless-spmf } (S\ (fst\ y))$   
 <proof>

**lemma**  $\text{set-spmf-S [simp]}: y \in \text{set-spmf } I \longrightarrow \text{set-spmf } (S\ (fst\ y)) = \text{range } (fst\ y)$   
 <proof>

**lemma**  $f\text{-inj-on}: y \in \text{set-spmf } I \longrightarrow \text{inj-on } (F\ (fst\ y))\ (\text{range } (fst\ y))$   
 <proof>

**lemma**  $\text{range-f}: y \in \text{set-spmf } I \longrightarrow x \in \text{range } (fst\ y) \longrightarrow F\ (fst\ y)\ x \in \text{range } (fst\ y)$   
 <proof>

**lemma**  $f\text{-inv-f [simp]}: y \in \text{set-spmf } I \longrightarrow x \in \text{range } (fst\ y) \longrightarrow F_{inv}\ (fst\ y)\ (snd\ y)\ (F\ (fst\ y)\ x) = x$   
 <proof>

**lemma**  $f\text{-inv-f' [simp]}: y \in \text{set-spmf } I \longrightarrow x \in \text{range } (fst\ y) \longrightarrow \text{Hilbert-Choice.inv-into } (\text{range } (fst\ y))\ (F\ (fst\ y))\ (F\ (fst\ y)\ x) = x$   
 <proof>

**lemma**  $B\text{-F-inv-rewrite}: (B\ \alpha\ (F_{inv}\ \alpha\ \tau\ y_\sigma')) = (B\ \alpha\ (F_{inv}\ \alpha\ \tau\ y_\sigma') = m1) = m1$   
 <proof>

**lemma** *uni-set-samp*:  
**assumes**  $y \in \text{set-spmf } I$   
**shows**  $\text{map-spmf } (\lambda x. F (\text{fst } y) x) (S (\text{fst } y)) = (S (\text{fst } y))$   
**(is ?lhs = ?rhs)**  
 $\langle \text{proof} \rangle$

We define the security property of the hard core predicate (HCP) using a game.

**definition** *HCP-game* ::  $('index, 'range) \text{advP2} \Rightarrow \text{bool} \Rightarrow \text{bool} \Rightarrow ('index, 'range) \text{dist2} \Rightarrow \text{bool} \text{spmf}$   
**where** *HCP-game*  $A = (\lambda \sigma b_\sigma D. \text{do} \{$   
 $(\alpha, \tau) \leftarrow I;$   
 $x \leftarrow S \alpha;$   
 $b' \leftarrow A \alpha \sigma b_\sigma D x;$   
 $\text{let } b = B \alpha (F_{inv} \alpha \tau x);$   
 $\text{return-spmf } (b = b') \}$

**definition** *HCP-adv*  $A \sigma b_\sigma D = |(\text{spmf } (HCP\text{-game } A \sigma b_\sigma D) \text{ True}) - 1/2|$

**end**

**end**

## 2.4 Oblivious transfer constructed from ETPs

Here we construct the OT protocol based on ETPs given in [5] (Chapter 4) and prove semi honest security for both parties. We show information theoretic security for Party 1 and reduce the security of Party 2 to the HCP assumption.

**theory** *ETP-OT imports*  
*HOL-Number-Theory.Cong*  
*ETP*  
*OT-Functionalities*  
*Semi-Honest-Def*  
**begin**

**type-synonym** *'range viewP1* =  $((\text{bool} \times \text{bool}) \times 'range \times 'range) \text{spmf}$   
**type-synonym** *'range dist1* =  $((\text{bool} \times \text{bool}) \times 'range \times 'range) \Rightarrow \text{bool} \text{spmf}$   
**type-synonym** *'index viewP2* =  $(\text{bool} \times 'index \times (\text{bool} \times \text{bool})) \text{spmf}$   
**type-synonym** *'index dist2* =  $(\text{bool} \times 'index \times \text{bool} \times \text{bool}) \Rightarrow \text{bool} \text{spmf}$   
**type-synonym**  $('index, 'range) \text{advP2} = 'index \Rightarrow \text{bool} \Rightarrow \text{bool} \Rightarrow 'index \text{dist2} \Rightarrow 'range \Rightarrow \text{bool} \text{spmf}$

**lemma** *if-False-True*:  $(\text{if } x \text{ then False else } \neg \text{False}) \longleftrightarrow (\text{if } x \text{ then False else True})$   
 $\langle \text{proof} \rangle$

**lemma** *if-then-True [simp]*:  $(\text{if } b \text{ then True else } x) \longleftrightarrow (\neg b \longrightarrow x)$   
 $\langle \text{proof} \rangle$

**lemma** *if-else-True* [*simp*]:  $(\text{if } b \text{ then } x \text{ else True}) \longleftrightarrow (b \longrightarrow x)$   
 ⟨*proof*⟩

**lemma** *inj-on-Not* [*simp*]: *inj-on Not A*  
 ⟨*proof*⟩

**locale** *ETP-base* = *etp*: *etp I domain range F F<sub>inv</sub> B*  
**for** *I* :: ('*index* × '*trap*) *spmf* — samples index and trapdoor  
**and** *domain* :: '*index* ⇒ '*range set*  
**and** *range* :: '*index* ⇒ '*range set*  
**and** *B* :: '*index* ⇒ '*range* ⇒ *bool* — hard core predicate  
**and** *F* :: '*index* ⇒ '*range* ⇒ '*range*  
**and** *F<sub>inv</sub>* :: '*index* ⇒ '*trap* ⇒ '*range* ⇒ '*range*  
**begin**

The probabilistic program that defines the protocol.

**definition** *protocol* :: (*bool* × *bool*) ⇒ *bool* ⇒ (*unit* × *bool*) *spmf*  
**where** *protocol input<sub>1</sub> σ* = *do* {  
*let* (*b<sub>σ</sub>*, *b<sub>σ'</sub>*) = *input<sub>1</sub>*;  
 (*α* :: '*index*, *τ* :: '*trap*) ← *I*;  
*x<sub>σ</sub>* :: '*range* ← *etp.S α*;  
*y<sub>σ'</sub>* :: '*range* ← *etp.S α*;  
*let* (*y<sub>σ</sub>* :: '*range*) = *F α x<sub>σ</sub>*;  
*let* (*x<sub>σ</sub>* :: '*range*) = *F<sub>inv</sub> α τ y<sub>σ</sub>*;  
*let* (*x<sub>σ'</sub>* :: '*range*) = *F<sub>inv</sub> α τ y<sub>σ'</sub>*;  
*let* (*β<sub>σ</sub>* :: *bool*) = *xor (B α x<sub>σ</sub>) b<sub>σ</sub>*;  
*let* (*β<sub>σ'</sub>* :: *bool*) = *xor (B α x<sub>σ'</sub>) b<sub>σ'</sub>*;  
*return-spmf* ((*λ* σ. *if* σ *then xor (B α x<sub>σ'</sub>) β<sub>σ'</sub> else xor (B α x<sub>σ</sub>) β<sub>σ</sub>))*

**lemma** *correctness*: *protocol (m0,m1) c* = *funct-OT-12 (m0,m1) c*  
 ⟨*proof*⟩

Party 1 views

**definition** *R1* :: (*bool* × *bool*) ⇒ *bool* ⇒ '*range viewP1*  
**where** *R1 input<sub>1</sub> σ* = *do* {  
*let* (*b<sub>0</sub>*, *b<sub>1</sub>*) = *input<sub>1</sub>*;  
 (*α*, *τ*) ← *I*;  
*x<sub>σ</sub>* ← *etp.S α*;  
*y<sub>σ'</sub>* ← *etp.S α*;  
*let* *y<sub>σ</sub>* = *F α x<sub>σ</sub>*;  
*return-spmf* ((*b<sub>0</sub>*, *b<sub>1</sub>*), *if* σ *then y<sub>σ'</sub> else y<sub>σ</sub>, if* σ *then y<sub>σ</sub> else y<sub>σ'</sub>*)

**lemma** *lossless-R1*: *lossless-spmf (R1 msgs σ)*  
 ⟨*proof*⟩

**definition** *S1* :: (*bool* × *bool*) ⇒ *unit* ⇒ '*range viewP1*  
**where** *S1* == (*λ input<sub>1</sub>* *λ* *σ*). *do* {  
*let* (*b<sub>0</sub>*, *b<sub>1</sub>*) = *input<sub>1</sub>*;

$(\alpha, \tau) \leftarrow I;$   
 $y_0 :: 'range \leftarrow \text{etp}.S \alpha;$   
 $y_1 \leftarrow \text{etp}.S \alpha;$   
 $\text{return-spmf } ((b_0, b_1), y_0, y_1))$

**lemma** *lossless-S1*: *lossless-spmf* (*S1 msgs* ())  
 $\langle \text{proof} \rangle$

Party 2 views

**definition** *R2* :: (*bool*  $\times$  *bool*)  $\Rightarrow$  *bool*  $\Rightarrow$  *'index viewP2*  
**where** *R2 msgs*  $\sigma = \text{do}$  {  
 $\text{let } (b_0, b_1) = \text{msgs};$   
 $(\alpha, \tau) \leftarrow I;$   
 $x_\sigma \leftarrow \text{etp}.S \alpha;$   
 $y_{\sigma'} \leftarrow \text{etp}.S \alpha;$   
 $\text{let } y_\sigma = F \alpha x_\sigma;$   
 $\text{let } x_\sigma = F_{\text{inv}} \alpha \tau y_\sigma;$   
 $\text{let } x_{\sigma'} = F_{\text{inv}} \alpha \tau y_{\sigma'};$   
 $\text{let } \beta_\sigma = (B \alpha x_\sigma) \oplus (\text{if } \sigma \text{ then } b_1 \text{ else } b_0);$   
 $\text{let } \beta_{\sigma'} = (B \alpha x_{\sigma'}) \oplus (\text{if } \sigma \text{ then } b_0 \text{ else } b_1);$   
 $\text{return-spmf } (\sigma, \alpha, (\beta_\sigma, \beta_{\sigma'}))$ }

**lemma** *lossless-R2*: *lossless-spmf* (*R2 msgs*  $\sigma$ )  
 $\langle \text{proof} \rangle$

**definition** *S2* :: *bool*  $\Rightarrow$  *bool*  $\Rightarrow$  *'index viewP2*  
**where** *S2*  $\sigma b_\sigma = \text{do}$  {  
 $(\alpha, \tau) \leftarrow I;$   
 $x_\sigma \leftarrow \text{etp}.S \alpha;$   
 $y_{\sigma'} \leftarrow \text{etp}.S \alpha;$   
 $\text{let } x_{\sigma'} = F_{\text{inv}} \alpha \tau y_{\sigma'};$   
 $\text{let } \beta_\sigma = (B \alpha x_\sigma) \oplus b_\sigma;$   
 $\text{let } \beta_{\sigma'} = B \alpha x_{\sigma'};$   
 $\text{return-spmf } (\sigma, \alpha, (\beta_\sigma, \beta_{\sigma'}))$ }

**lemma** *lossless-S2*: *lossless-spmf* (*S2*  $\sigma b_\sigma$ )  
 $\langle \text{proof} \rangle$

Security for Party 1

We have information theoretic security for Party 1.

**lemma** *P1-security*: *R1 input*<sub>1</sub>  $\sigma = \text{funct-OT-12 } x y \gg= (\lambda (s_1, s_2). S1 \text{ input}_1 s_1)$

**including** *monad-normalisation*  
 $\langle \text{proof} \rangle$

The adversary used in proof of security for party 2

**definition** *A* :: (*'index*, *'range*) *advP2*  
**where** *A*  $\alpha \sigma b_\sigma D_2 x = \text{do}$  {

$\beta_{\sigma'} \leftarrow \text{coin-spmf};$   
 $x_{\sigma} \leftarrow \text{etp.S } \alpha;$   
 $\text{let } \beta_{\sigma} = (B \ \alpha \ x_{\sigma}) \oplus b_{\sigma};$   
 $d \leftarrow D2(\sigma, \alpha, \beta_{\sigma}, \beta_{\sigma}');$   
 $\text{return-spmf}(\text{if } d \text{ then } \beta_{\sigma'} \text{ else } \neg \beta_{\sigma}')\}$

**lemma** *lossless-A:*

**assumes**  $\forall \text{ view. lossless-spmf } (D2 \ \text{view})$   
**shows**  $y \in \text{set-spmf } I \longrightarrow \text{lossless-spmf } (\mathcal{A} \ (\text{fst } y) \ \sigma \ b_{\sigma} \ D2 \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *assm-bound-funct-OT-12:*

**assumes**  $\text{etp.HCP-adv } \mathcal{A} \ \sigma \ (\text{if } \sigma \ \text{then } b1 \ \text{else } b0) \ D \leq \text{HCP-ad}$   
**shows**  $|\text{spmf } (\text{funct-OT-12 } (b0, b1) \ \sigma \ggg (\lambda \ (out1, out2). \text{etp.HCP-game } \mathcal{A} \ \sigma \ out2 \ D))) \ \text{True} - 1/2| \leq \text{HCP-ad}$   
 $(\text{is } ?lhs \leq \text{HCP-ad})$   
 $\langle \text{proof} \rangle$

**lemma** *assm-bound-funct-OT-12-collapse:*

**assumes**  $\forall b_{\sigma}. \text{etp.HCP-adv } \mathcal{A} \ \sigma \ b_{\sigma} \ D \leq \text{HCP-ad}$   
**shows**  $|\text{spmf } (\text{funct-OT-12 } m1 \ \sigma \ggg (\lambda \ (out1, out2). \text{etp.HCP-game } \mathcal{A} \ \sigma \ out2 \ D))) \ \text{True} - 1/2| \leq \text{HCP-ad}$   
 $\langle \text{proof} \rangle$

To prove security for party 2 we split the proof on the cases on party 2's input

**lemma** *R2-S2-False:*

**assumes**  $((\text{if } \sigma \ \text{then } b0 \ \text{else } b1) = \text{False})$   
**shows**  $\text{spmf } (R2 \ (b0, b1) \ \sigma \ggg (D2 \ :: \ (\text{bool} \times \ 'index \times \ \text{bool} \times \ \text{bool}) \Rightarrow \ \text{bool} \ \text{spmf})) \ \text{True}$   
 $= \text{spmf } (\text{funct-OT-12 } (b0, b1) \ \sigma \ggg (\lambda \ (out1, out2). S2 \ \sigma \ out2 \ggg D2)) \ \text{True}$   
 $\langle \text{proof} \rangle$

**lemma** *R2-S2-True:*

**assumes**  $((\text{if } \sigma \ \text{then } b0 \ \text{else } b1) = \text{True})$   
**and**  $\text{lossless-D: } \forall a. \text{lossless-spmf } (D2 \ a)$   
**shows**  $|\text{spmf } (\text{bind-spmf } (R2 \ (b0, b1) \ \sigma) \ D2) \ \text{True}) - \text{spmf } (\text{funct-OT-12 } (b0, b1) \ \sigma \ggg (\lambda \ (out1, out2). S2 \ \sigma \ out2 \ggg (\lambda \ \text{view. } D2 \ \text{view}))) \ \text{True}|$   
 $= |2 * (\text{spmf } (\text{etp.HCP-game } \mathcal{A} \ \sigma \ (\text{if } \sigma \ \text{then } b1 \ \text{else } b0) \ D2) \ \text{True}) - 1/2|$   
 $\langle \text{proof} \rangle$   
**including** *monad-normalisation*  
 $\langle \text{proof} \rangle$

**lemma** *P2-adv-bound:*

**assumes**  $\text{lossless-D: } \forall a. \text{lossless-spmf } (D2 \ a)$   
**shows**  $|\text{spmf } (\text{bind-spmf } (R2 \ (b0, b1) \ \sigma) \ D2) \ \text{True}) - \text{spmf } (\text{funct-OT-12 } (b0, b1) \ \sigma \ggg (\lambda \ (out1, out2). S2 \ \sigma \ out2 \ggg (\lambda \ \text{view. } D2 \ \text{view}))) \ \text{True}|$



$\leq |2 * ((\text{spmf } (\text{etp.HCP-game } \mathcal{A} \sigma \text{ (if } \sigma \text{ then } b1 \text{ else } b0) D2))$   
 $\text{True}) - 1/2)|$   
 <proof>

**sublocale** *OT-12: sim-det-def R1 S1 R2 S2 funct-OT-12 protocol*  
 <proof>

**lemma** *correct: OT-12.correctness m1 m2*  
 <proof>

**lemma** *P1-security-inf-the: OT-12.perfect-sec-P1 m1 m2*  
 <proof>

**lemma** *P2-security:*  
**assumes**  $\forall a. \text{lossless-spmf } (D a)$   
**and**  $\forall b_\sigma. \text{etp.HCP-adv } \mathcal{A} m2 b_\sigma D \leq \text{HCP-ad}$   
**shows**  $\text{OT-12.adv-P2 } m1 m2 D \leq 2 * \text{HCP-ad}$   
 <proof>

**end**

We also consider the asymptotic case for security proofs

**locale** *ETP-sec-para =*  
**fixes**  $I :: \text{nat} \Rightarrow ('index \times 'trap) \text{ spmf}$   
**and**  $\text{domain} :: 'index \Rightarrow 'range \text{ set}$   
**and**  $\text{range} :: 'index \Rightarrow 'range \text{ set}$   
**and**  $f :: 'index \Rightarrow ('range \Rightarrow 'range)$   
**and**  $F :: 'index \Rightarrow 'range \Rightarrow 'range$   
**and**  $F_{inv} :: 'index \Rightarrow 'trap \Rightarrow 'range \Rightarrow 'range$   
**and**  $B :: 'index \Rightarrow 'range \Rightarrow \text{bool}$   
**assumes** *ETP-base:  $\bigwedge n. \text{ETP-base } (I n) \text{ domain range } F F_{inv}$*   
**begin**

**sublocale** *ETP-base (I n) domain range*  
 <proof>

**lemma** *correct-asym: OT-12.correctness n m1 m2*  
 <proof>

**lemma** *P1-sec-asym: OT-12.perfect-sec-P1 n m1 m2*  
 <proof>

**lemma** *P2-sec-asym:*  
**assumes**  $\forall a. \text{lossless-spmf } (D a)$   
**and** *HCP-adv-neg: negligible ( $\lambda n. \text{etp-advantage } n$ )*  
**and** *etp-adv-bound:  $\forall b_\sigma n. \text{etp.HCP-adv } n \mathcal{A} m2 b_\sigma D \leq \text{etp-advantage } n$*   
**shows** *negligible ( $\lambda n. \text{OT-12.adv-P2 } n m1 m2 D$ )*  
 <proof>

**end**

**end**

### 2.4.1 RSA instantiation

It is known that the RSA collection forms an ETP. Here we instantiate our proof of security for OT that uses a general ETP for RSA. We use the proof of the general construction of OT. The main proof effort here is in showing the RSA collection meets the requirements of an ETP, mainly this involves showing the RSA mapping is a bijection.

**theory** *ETP-RSA-OT* **imports**

*ETP-OT*

*Number-Theory-Aux*

*Uniform-Sampling*

**begin**

**type-synonym** *index* = (*nat* × *nat*)

**type-synonym** *trap* = *nat*

**type-synonym** *range* = *nat*

**type-synonym** *domain* = *nat*

**type-synonym** *viewP1* = ((*bool* × *bool*) × *nat* × *nat*) *spmf*

**type-synonym** *viewP2* = (*bool* × *index* × (*bool* × *bool*)) *spmf*

**type-synonym** *dist2* = (*bool* × *index* × *bool* × *bool*) ⇒ *bool* *spmf*

**type-synonym** *advP2* = *index* ⇒ *bool* ⇒ *bool* ⇒ *dist2* ⇒ *bool* *spmf*

**locale** *rsa-base* =

**fixes** *prime-set* :: *nat set* — the set of primes used

**and** *B* :: *index* ⇒ *nat* ⇒ *bool*

**assumes** *prime-set-ass*: *prime-set* ⊆ {*x. prime x* ∧ *x* > 2}

**and** *finite-prime-set*: *finite prime-set*

**and** *prime-set-gt-2*: *card prime-set* > 2

**begin**

**lemma** *prime-set-non-empty*: *prime-set* ≠ {}

⟨*proof*⟩

**definition** *coprime-set* :: *nat* ⇒ *nat set*

**where** *coprime-set* *N* ≡ {*x. coprime x N* ∧ *x* > 1 ∧ *x* < *N*}

**lemma** *coprime-set-non-empty*:

**assumes** *N* > 2

**shows** *coprime-set N* ≠ {}

⟨*proof*⟩

**definition** *sample-coprime* :: *nat* ⇒ *nat spmf*

**where** *sample-coprime N* = *spmf-of-set (coprime-set (N))*

**lemma** *sample-coprime-e-gt-1*:  
**assumes**  $e \in \text{set-spmf } (\text{sample-coprime } N)$   
**shows**  $e > 1$   
 $\langle \text{proof} \rangle$

**lemma** *lossless-sample-coprime*:  
**assumes**  $\neg \text{prime } N$   
**and**  $N > 2$   
**shows**  $\text{lossless-spmf } (\text{sample-coprime } N)$   
 $\langle \text{proof} \rangle$

**lemma** *set-spmf-sample-coprime*:  
**shows**  $\text{set-spmf } (\text{sample-coprime } N) = \{x. \text{coprime } x \ N \wedge x > 1 \wedge x < N\}$   
 $\langle \text{proof} \rangle$

**definition** *sample-primes* ::  $\text{nat spmf}$   
**where**  $\text{sample-primes} = \text{spmf-of-set prime-set}$

**lemma** *lossless-sample-primes*:  
**shows**  $\text{lossless-spmf sample-primes}$   
 $\langle \text{proof} \rangle$

**lemma** *set-spmf-sample-primes*:  
**shows**  $\text{set-spmf sample-primes} \subseteq \{x. \text{prime } x \wedge x > 2\}$   
 $\langle \text{proof} \rangle$

**lemma** *mem-samp-primes-gt-2*:  
**shows**  $x \in \text{set-spmf sample-primes} \implies x > 2$   
 $\langle \text{proof} \rangle$

**lemma** *mem-samp-primes-prime*:  
**shows**  $x \in \text{set-spmf sample-primes} \implies \text{prime } x$   
 $\langle \text{proof} \rangle$

**definition** *sample-primes-excl* ::  $\text{nat set} \Rightarrow \text{nat spmf}$   
**where**  $\text{sample-primes-excl } P = \text{spmf-of-set } (\text{prime-set} - P)$

**lemma** *lossless-sample-primes-excl*:  
**shows**  $\text{lossless-spmf } (\text{sample-primes-excl } \{P\})$   
 $\langle \text{proof} \rangle$

**definition** *sample-set-excl* ::  $\text{nat set} \Rightarrow \text{nat set} \Rightarrow \text{nat spmf}$   
**where**  $\text{sample-set-excl } Q \ P = \text{spmf-of-set } (Q - P)$

**lemma** *set-spmf-sample-set-excl* [*simp*]:  
**assumes**  $\text{finite } (Q - P)$   
**shows**  $\text{set-spmf } (\text{sample-set-excl } Q \ P) = (Q - P)$   
 $\langle \text{proof} \rangle$

**lemma** *lossless-sample-set-excl*:

**assumes** *finite Q*  
**and** *card Q > 2*  
**shows** *lossless-spmf (sample-set-excl Q {P})*  
*<proof>*

**lemma** *mem-samp-primes-excl-gt-2*:

**shows** *x ∈ set-spmf (sample-set-excl prime-set {y}) ⇒ x > 2*  
*<proof>*

**lemma** *mem-samp-primes-excl-prime* :

**shows** *x ∈ set-spmf (sample-set-excl prime-set {y}) ⇒ prime x*  
*<proof>*

**lemma** *sample-coprime-lem*:

**assumes** *x ∈ set-spmf sample-primes*  
**and** *y ∈ set-spmf (sample-set-excl prime-set {x})*  
**shows** *lossless-spmf (sample-coprime ((x - Suc 0) \* (y - Suc 0)))*  
*<proof>*

**definition** *I :: (index × trap) spmf*

**where** *I = do* {  
  *P ← sample-primes;*  
  *Q ← sample-set-excl prime-set {P};*  
  *let N = P \* Q;*  
  *let N' = (P - 1) \* (Q - 1);*  
  *e ← sample-coprime N';*  
  *let d = nat ((fst (bezw e N')) mod N');*  
  *return-spmf ((N, e), d)}*

**lemma** *lossless-I: lossless-spmf I*

*<proof>*

**lemma** *set-spmf-I-N*:

**assumes** *((N, e), d) ∈ set-spmf I*  
**obtains** *P Q* **where** *N = P \* Q*  
**and** *P ≠ Q*  
**and** *prime P*  
**and** *prime Q*  
**and** *coprime e ((P - 1) \* (Q - 1))*  
**and** *d = nat (fst (bezw e ((P - 1) \* (Q - 1))) mod int ((P - 1) \* (Q - 1)))*  
*<proof>*

**lemma** *set-spmf-I-e-d*:

**assumes** *((N, e), d) ∈ set-spmf I*  
**shows** *e > 1 and d > 1*  
*<proof>*

**definition** *domain :: index ⇒ nat set*

**where**  $domain\ index \equiv \{..< fst\ index\}$

**definition**  $range :: index \Rightarrow nat\ set$   
**where**  $range\ index \equiv \{..< fst\ index\}$

**lemma**  $finite-range: finite\ (range\ index)$   
 $\langle proof \rangle$

**lemma**  $dom-eq-ran: domain\ index = range\ index$   
 $\langle proof \rangle$

**definition**  $F :: index \Rightarrow (nat \Rightarrow nat)$   
**where**  $F\ index\ x = x \wedge^{(snd\ index)}\ mod\ (fst\ index)$

**definition**  $F_{inv} :: index \Rightarrow trap \Rightarrow nat \Rightarrow nat$   
**where**  $F_{inv}\ \alpha\ \tau\ y = y \wedge^{\tau}\ mod\ (fst\ \alpha)$

We must prove the RSA function is a bijection

**lemma**  $rsa-bijection:$

**assumes**  $coprime: coprime\ e\ ((P-1)*(Q-1))$   
**and**  $prime-P: prime\ (P::nat)$   
**and**  $prime-Q: prime\ Q$   
**and**  $P-neq-Q: P \neq Q$   
**and**  $x-lt-pq: x < P * Q$   
**and**  $y-lt-pd: y < P * Q$   
**and**  $rsa-map-eq: x \wedge^e\ mod\ (P * Q) = y \wedge^e\ mod\ (P * Q)$   
**shows**  $x = y$   
 $\langle proof \rangle$

**lemma**  $rsa-bij-betw:$

**assumes**  $coprime\ e\ ((P - 1)*(Q - 1))$   
**and**  $prime\ P$   
**and**  $prime\ Q$   
**and**  $P \neq Q$   
**shows**  $bij-betw\ (F\ ((P * Q),\ e))\ (range\ ((P * Q),\ e))\ (range\ ((P * Q),\ e))$   
 $\langle proof \rangle$

**lemma**  $bij-betw1:$

**assumes**  $((N,e),d) \in set-spmf\ I$   
**shows**  $bij-betw\ (F\ ((N),\ e))\ (range\ ((N),\ e))\ (range\ ((N),\ e))$   
 $\langle proof \rangle$

**lemma**

**assumes**  $P\ dvd\ x$   
**shows**  $[x = 0] (mod\ P)$   
 $\langle proof \rangle$

**lemma**  $rsa-inv:$

**assumes**  $d: d = nat\ (fst\ (bezw\ e\ ((P-1)*(Q-1))))\ mod\ int\ ((P-1)*(Q-1))$

```

and coprime: coprime e ((P-1)*(Q-1))
and prime-P: prime (P::nat)
and prime-Q: prime Q
and P-neq-Q: P ≠ Q
and e-gt-1: e > 1
and d-gt-1: d > 1
shows (((x) ^ e) mod (P*Q)) ^ d) mod (P*Q) = x mod (P*Q)
⟨proof⟩

```

```

lemma rsa-inv-set-spmf-I:
  assumes ((N, e), d) ∈ set-spmf I
  shows (((x::nat) ^ e) mod N) ^ d) mod N = x mod N
⟨proof⟩

```

```

sublocale etp-rsa: etp I domain range F Finv
⟨proof⟩

```

```

sublocale etp: ETP-base I domain range B F Finv
⟨proof⟩

```

After proving the RSA collection is an ETP the proofs of security come easily from the general proofs.

```

lemma correctness-rsa: etp.OT-12.correctness m1 m2
⟨proof⟩

```

```

lemma P1-security-rsa: etp.OT-12.perfect-sec-P1 m1 m2
⟨proof⟩

```

```

lemma P2-security-rsa:
  assumes ∀ a. lossless-spmf (D a)
  and ∧ bσ. local.etp-rsa.HCP-adv etp.ℳ m2 bσ D ≤ HCP-ad
  shows etp.OT-12.adv-P2 m1 m2 D ≤ 2 * HCP-ad
⟨proof⟩

```

**end**

```

locale rsa-asym =
  fixes prime-set :: nat ⇒ nat set
  and B :: index ⇒ nat ⇒ bool
  assumes rsa-proof-assm: ∧ n. rsa-base (prime-set n)
begin

```

```

sublocale rsa-base (prime-set n) B
⟨proof⟩

```

```

lemma correctness-rsa-asym:
  shows etp.OT-12.correctness n m1 m2
⟨proof⟩

```

**lemma** *P1-sec-asymp*: *etp.OT-12.perfect-sec-P1*  $n$   $m1$   $m2$   
 ⟨*proof*⟩

**lemma** *P2-sec-asymp*:  
**assumes**  $\forall a$ . *lossless-spmf* ( $D$   $a$ )  
**and** *HCP-adv-neg*: *negligible* ( $\lambda n$ . *hcp-advantage*  $n$ )  
**and** *hcp-adv-bound*:  $\forall b_\sigma n$ . *local.etp-rsa.HCP-adv*  $n$  *etp.A*  $m2$   $b_\sigma$   $D \leq$  *hcp-advantage*  
 $n$   
**shows** *negligible* ( $\lambda n$ . *etp.OT-12.adv-P2*  $n$   $m1$   $m2$   $D$ )  
 ⟨*proof*⟩

**end**

**end**

## 2.5 Noar Pinkas OT

Here we prove security for the Noar Pinkas OT from [7].

**theory** *Noar-Pinkas-OT* **imports**

*Cyclic-Group-Ext*  
*Game-Based-Crypto.Diffie-Hellman*  
*OT-Functionalities*  
*Semi-Honest-Def*  
*Uniform-Sampling*

**begin**

**locale** *np-base* =  
**fixes**  $\mathcal{G} :: 'grp$  *cyclic-group* (**structure**)  
**assumes** *finite-group*: *finite* (*carrier*  $\mathcal{G}$ )  
**and** *or-gt-0*:  $0 <$  *order*  $\mathcal{G}$   
**and** *prime-order*: *prime* (*order*  $\mathcal{G}$ )  
**begin**

**lemma** *prime-field*:  $a <$  (*order*  $\mathcal{G}$ )  $\implies a \neq 0 \implies$  *coprime*  $a$  (*order*  $\mathcal{G}$ )  
 ⟨*proof*⟩

**lemma** *weight-sample-uniform-units*: *weight-spmf* (*sample-uniform-units* (*order*  $\mathcal{G}$ ))  
 =  $1$   
 ⟨*proof*⟩

**definition** *protocol* :: ( $'grp \times 'grp$ )  $\implies$  *bool*  $\implies$  (*unit*  $\times$   $'grp$ ) *spmf*  
**where** *protocol*  $M$   $v =$  *do* {  
*let* ( $m0, m1$ ) =  $M$ ;  
 $a :: nat \leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
 $b :: nat \leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*let*  $c_v = (a * b) \bmod$  (*order*  $\mathcal{G}$ );  
 $c_v' :: nat \leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
 $r0 :: nat \leftarrow$  *sample-uniform-units* (*order*  $\mathcal{G}$ );

```

s0 :: nat ← sample-uniform-units (order G);
let w0 = (g [∧] a) [∧] s0 ⊗ g [∧] r0;
let z0' = ((g [∧] (if v then c_v' else c_v)) [∧] s0) ⊗ ((g [∧] b) [∧] r0);
r1 :: nat ← sample-uniform-units (order G);
s1 :: nat ← sample-uniform-units (order G);
let w1 = (g [∧] a) [∧] s1 ⊗ g [∧] r1;
let z1' = ((g [∧] ((if v then c_v else c_v'))) [∧] s1) ⊗ ((g [∧] b) [∧] r1);
let enc-m0 = z0' ⊗ m0;
let enc-m1 = z1' ⊗ m1;
let out-2 = (if v then enc-m1 ⊗ inv (w1 [∧] b) else enc-m0 ⊗ inv (w0 [∧] b));
return-spmf (((), out-2)}

```

**lemma** *lossless-protocol*: *lossless-spmf* (protocol  $M$   $\sigma$ )  
 ⟨proof⟩

**type-synonym** *'grp' view1* = (('grp' × 'grp') × ('grp' × 'grp' × 'grp' × 'grp'))  
 spmf

**type-synonym** *'grp' dist-adversary* = (('grp' × 'grp') × 'grp' × 'grp' × 'grp' × 'grp')  
 ⇒ bool spmf

**definition** *R1* :: ('grp × 'grp) ⇒ bool ⇒ 'grp view1  
 where *R1* msgs  $\sigma$  = do {  
 let (m0, m1) = msgs;  
 a ← sample-uniform (order G);  
 b ← sample-uniform (order G);  
 let c $_{\sigma}$  = a\*b;  
 c' $_{\sigma}$  ← sample-uniform (order G);  
 return-spmf (msgs, (g [∧] a, g [∧] b, (if  $\sigma$  then g [∧] c' $_{\sigma}$  else g [∧] c $_{\sigma}$ ), (if  $\sigma$   
 then g [∧] c $_{\sigma}$  else g [∧] c' $_{\sigma}$ )))}

**lemma** *lossless-R1*: *lossless-spmf* (*R1*  $M$   $\sigma$ )  
 ⟨proof⟩

**definition** *inter* :: ('grp × 'grp) ⇒ 'grp view1  
 where *inter* msgs = do {  
 a ← sample-uniform (order G);  
 b ← sample-uniform (order G);  
 c ← sample-uniform (order G);  
 d ← sample-uniform (order G);  
 return-spmf (msgs, g [∧] a, g [∧] b, g [∧] c, g [∧] d)}

**definition** *S1* :: ('grp × 'grp) ⇒ unit ⇒ 'grp view1  
 where *S1* msgs out1 = do {  
 let (m0, m1) = msgs;  
 a ← sample-uniform (order G);  
 b ← sample-uniform (order G);  
 c ← sample-uniform (order G);  
 return-spmf (msgs, (g [∧] a, g [∧] b, g [∧] c, g [∧] (a\*b)))}



**lemma** *lossless-S1: lossless-spmf (S1 M out1)*

*<proof>*

**fun** *R1-inter-adversary* :: 'grp dist-adversary  $\Rightarrow$  ('grp  $\times$  'grp)  $\Rightarrow$  'grp  $\Rightarrow$  'grp  $\Rightarrow$  'grp  $\Rightarrow$  bool spmf

**where** *R1-inter-adversary*  $\mathcal{A}$  msgs  $\alpha$   $\beta$   $\gamma$  = do {  
 $c \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $\mathcal{A}$  (msgs,  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\mathbf{g} [\uparrow] c$ )}

**fun** *inter-S1-adversary* :: 'grp dist-adversary  $\Rightarrow$  ('grp  $\times$  'grp)  $\Rightarrow$  'grp  $\Rightarrow$  'grp  $\Rightarrow$  'grp  $\Rightarrow$  bool spmf

**where** *inter-S1-adversary*  $\mathcal{A}$  msgs  $\alpha$   $\beta$   $\gamma$  = do {  
 $c \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $\mathcal{A}$  (msgs,  $\alpha$ ,  $\beta$ ,  $\mathbf{g} [\uparrow] c$ ,  $\gamma$ )}

**sublocale** *ddh*: ddh  $\mathcal{G}$  *<proof>*

**definition** *R2* :: ('grp  $\times$  'grp)  $\Rightarrow$  bool  $\Rightarrow$  (bool  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp) spmf

**where** *R2*  $M$   $v$  = do {  
 $let$  ( $m0, m1$ ) =  $M$ ;  
 $a :: nat \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $b :: nat \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $let$   $c_v = (a*b) \bmod$  (order  $\mathcal{G}$ );  
 $c_v' :: nat \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $r0 :: nat \leftarrow$  sample-uniform-units (order  $\mathcal{G}$ );  
 $s0 :: nat \leftarrow$  sample-uniform-units (order  $\mathcal{G}$ );  
 $let$   $w0 = (\mathbf{g} [\uparrow] a) [\uparrow] s0 \otimes \mathbf{g} [\uparrow] r0$ ;  
 $let$   $z = ((\mathbf{g} [\uparrow] c_v') [\uparrow] s0) \otimes ((\mathbf{g} [\uparrow] b) [\uparrow] r0)$ ;  
 $r1 :: nat \leftarrow$  sample-uniform-units (order  $\mathcal{G}$ );  
 $s1 :: nat \leftarrow$  sample-uniform-units (order  $\mathcal{G}$ );  
 $let$   $w1 = (\mathbf{g} [\uparrow] a) [\uparrow] s1 \otimes \mathbf{g} [\uparrow] r1$ ;  
 $let$   $z' = ((\mathbf{g} [\uparrow] (c_v)) [\uparrow] s1) \otimes ((\mathbf{g} [\uparrow] b) [\uparrow] r1)$ ;  
 $let$   $enc-m = z \otimes$  (if  $v$  then  $m0$  else  $m1$ );  
 $let$   $enc-m' = z' \otimes$  (if  $v$  then  $m1$  else  $m0$ );  
 $return$ -spmf( $v$ ,  $\mathbf{g} [\uparrow] a$ ,  $\mathbf{g} [\uparrow] b$ ,  $\mathbf{g} [\uparrow] c_v$ ,  $w0$ ,  $enc-m$ ,  $w1$ ,  $enc-m'$ )}

**lemma** *lossless-R2: lossless-spmf (R2 M  $\sigma$ )*

*<proof>*

**definition** *S2* :: bool  $\Rightarrow$  'grp  $\Rightarrow$  (bool  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp  $\times$  'grp) spmf

**where** *S2*  $v$   $m$  = do {  
 $a :: nat \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $b :: nat \leftarrow$  sample-uniform (order  $\mathcal{G}$ );  
 $let$   $c_v = (a*b) \bmod$  (order  $\mathcal{G}$ );  
 $r0 :: nat \leftarrow$  sample-uniform-units (order  $\mathcal{G}$ );  
 $s0 :: nat \leftarrow$  sample-uniform-units (order  $\mathcal{G}$ );

```

let w0 = (g [∧] a) [∧] s0 ⊗ g [∧] r0;
r1 :: nat ← sample-uniform-units (order G);
s1 :: nat ← sample-uniform-units (order G);
let w1 = (g [∧] a) [∧] s1 ⊗ g [∧] r1;
let z' = ((g [∧] (c_v)) [∧] s1) ⊗ ((g [∧] b) [∧] r1);
s' ← sample-uniform (order G);
let enc-m = g [∧] s';
let enc-m' = z' ⊗ m ;
return-spmf(v, g [∧] a, g [∧] b, g [∧] c_v, w0, enc-m, w1, enc-m')

```

**lemma** *lossless-S2*: *lossless-spmf (S2 σ out2)*  
 ⟨proof⟩

**sublocale** *sim-def*: *sim-det-def R1 S1 R2 S2 funct-OT-12 protocol*  
 ⟨proof⟩

**end**

**locale** *np* = *np-base* + *cyclic-group G*  
**begin**

**lemma** *protocol-inverse*:

```

  assumes m0 ∈ carrier G m1 ∈ carrier G
  shows ((g [∧] ((a*b) mod (order G))) [∧] (s1 :: nat)) ⊗ ((g [∧] b) [∧] (r1::nat))
  ⊗ (if v then m0 else m1) ⊗ inv (((g [∧] a) [∧] s1) ⊗ g [∧] r1) [∧] b)
    = (if v then m0 else m1)
  (is ?lhs = ?rhs)
  ⟨proof⟩

```

**lemma** *correctness*:

```

  assumes m0 ∈ carrier G m1 ∈ carrier G
  shows sim-def.correctness (m0,m1) σ
  ⟨proof⟩

```

**lemma** *security-P1*:

```

  shows sim-def.adv-P1 msgs σ D ≤ ddh.advantage (R1-inter-adversary D msgs)
  + ddh.advantage (inter-S1-adversary D msgs)
  (is ?lhs ≤ ?rhs)
  ⟨proof⟩ including monad-normalisation
  ⟨proof⟩

```

**lemma** *add-mult-one-time-pad*:

```

  assumes s0 < order G
  and s0 ≠ 0
  shows map-spmf (λ c_v'. (((b* r0) + (s0 * c_v')) mod(order G))) (sample-uniform
  (order G)) = sample-uniform (order G)
  ⟨proof⟩

```

**lemma** *security-P2*:

```

assumes  $m0 \in \text{carrier } \mathcal{G} \ m1 \in \text{carrier } \mathcal{G}$ 
shows  $\text{sim-def.perfect-sec-P2 } (m0, m1) \ \sigma$ 
<proof>
  including monad-normalisation
  <proof>

end

locale np-asymp =
  fixes  $\mathcal{G} :: \text{security} \Rightarrow \text{'grp cyclic-group}$ 
  assumes  $\text{np: } \bigwedge \eta. \text{np } (\mathcal{G} \ \eta)$ 
begin

sublocale  $\text{np } \mathcal{G} \ \eta$  for  $\eta$  <proof>

theorem correctness-asymp:
  assumes  $m0 \in \text{carrier } (\mathcal{G} \ \eta) \ m1 \in \text{carrier } (\mathcal{G} \ \eta)$ 
  shows  $\text{sim-def.correctness } \eta \ (m0, m1) \ \sigma$ 
  <proof>

theorem security-P1-asymp:
  assumes  $\text{negligible } (\lambda \eta. \text{ddh.advantage } \eta \ (\text{inter-S1-adversary } \eta \ D \ \text{msgs}))$ 
  and  $\text{negligible } (\lambda \eta. \text{ddh.advantage } \eta \ (\text{R1-inter-adversary } \eta \ D \ \text{msgs}))$ 
  shows  $\text{negligible } (\lambda \eta. \text{sim-def.adv-P1 } \eta \ \text{msgs} \ \sigma \ D)$ 
  <proof>

theorem security-P2-asymp:
  assumes  $m0 \in \text{carrier } (\mathcal{G} \ \eta) \ m1 \in \text{carrier } (\mathcal{G} \ \eta)$ 
  shows  $\text{sim-def.perfect-sec-P2 } \eta \ (m0, m1) \ \sigma$ 
  <proof>

end

end

```

## 2.6 1-out-of-2 OT to 1-out-of-4 OT

Here we construct a protocol that achieves 1-out-of-4 OT from 1-out-of-2 OT. We follow the protocol for constructing 1-out-of-n OT from 1-out-of-2 OT from [2]. We assume the security properties on 1-out-of-2 OT.

```

theory OT14 imports
  Semi-Honest-Def
  OT-Functionalities
  Uniform-Sampling
begin

type-synonym input1 =  $\text{bool} \times \text{bool} \times \text{bool} \times \text{bool}$ 
type-synonym input2 =  $\text{bool} \times \text{bool}$ 

```

**type-synonym**  $'v\text{-}OT121'$   $view1 = (input1 \times (bool \times bool \times bool \times bool \times bool \times bool)) \times 'v\text{-}OT121' \times 'v\text{-}OT121' \times 'v\text{-}OT121'$   
**type-synonym**  $'v\text{-}OT122'$   $view2 = (input2 \times (bool \times bool \times bool \times bool)) \times 'v\text{-}OT122' \times 'v\text{-}OT122' \times 'v\text{-}OT122'$

**locale**  $ot14\text{-}base =$   
**fixes**  $S1\text{-}OT12 :: (bool \times bool) \Rightarrow unit \Rightarrow 'v\text{-}OT121$  *spmf* — simulator for party 1 in OT12  
**and**  $R1\text{-}OT12 :: (bool \times bool) \Rightarrow bool \Rightarrow 'v\text{-}OT121$  *spmf* — real view for party 1 in OT12  
**and**  $adv\text{-}OT12 :: real$   
**and**  $S2\text{-}OT12 :: bool \Rightarrow bool \Rightarrow 'v\text{-}OT122$  *spmf*  
**and**  $R2\text{-}OT12 :: (bool \times bool) \Rightarrow bool \Rightarrow 'v\text{-}OT122$  *spmf*  
**and**  $protocol\text{-}OT12 :: (bool \times bool) \Rightarrow bool \Rightarrow (unit \times bool)$  *spmf*  
**assumes**  $ass\text{-}adv\text{-}OT12: sim\text{-}det\text{-}def.adv\text{-}P1$   $R1\text{-}OT12$   $S1\text{-}OT12$   $funct\text{-}OT12$   $(m0,m1)$   
 $c$   $D \leq adv\text{-}OT12$  — bound the advantage of OT12 for party 1  
**and**  $inf\text{-}th\text{-}OT12\text{-}P2: sim\text{-}det\text{-}def.perfect\text{-}sec\text{-}P2$   $R2\text{-}OT12$   $S2\text{-}OT12$   $funct\text{-}OT12$   
 $(m0,m1)$   $\sigma$  — information theoretic security for party 2  
**and**  $correct: protocol\text{-}OT12$   $msgs$   $b = funct\text{-}OT\text{-}12$   $msgs$   $b$   
**and**  $lossless\text{-}R1\text{-}12: lossless\text{-}spmf$   $(R1\text{-}OT12$   $m$   $c)$   
**and**  $lossless\text{-}S1\text{-}12: lossless\text{-}spmf$   $(S1\text{-}OT12$   $m$   $out1)$   
**and**  $lossless\text{-}S2\text{-}12: lossless\text{-}spmf$   $(S2\text{-}OT12$   $c$   $out2)$   
**and**  $lossless\text{-}R2\text{-}12: lossless\text{-}spmf$   $(R2\text{-}OT12$   $M$   $c)$   
**and**  $lossless\text{-}funct\text{-}OT12: lossless\text{-}spmf$   $(funct\text{-}OT12$   $(m0,m1)$   $c)$   
**and**  $lossless\text{-}protocol\text{-}OT12: lossless\text{-}spmf$   $(protocol\text{-}OT12$   $M$   $C)$   
**begin**

**sublocale**  $OT\text{-}12\text{-}sim: sim\text{-}det\text{-}def$   $R1\text{-}OT12$   $S1\text{-}OT12$   $R2\text{-}OT12$   $S2\text{-}OT12$   $funct\text{-}OT\text{-}12$   
 $protocol\text{-}OT12$   
 $\langle proof \rangle$

**lemma**  $OT\text{-}12\text{-}P1\text{-}assms\text{-}bound'$ :  $|spmf$   $(bind\text{-}spmf$   $(R1\text{-}OT12$   $(m0,m1)$   $c)$   $(\lambda view.$   
 $((D::'v\text{-}OT121 \Rightarrow bool$  *spmf*)  $view$  )))  $True$   
 $- spmf$   $(bind\text{-}spmf$   $(S1\text{-}OT12$   $(m0,m1)$   $()$ )  $(\lambda view.$   $(D$   $view$  )))  $True$   
 $\leq adv\text{-}OT12$   
 $\langle proof \rangle$

**lemma**  $OT\text{-}12\text{-}P2\text{-}assm$ :  $R2\text{-}OT12$   $(m0,m1)$   $\sigma = funct\text{-}OT\text{-}12$   $(m0,m1)$   $\sigma \gg= (\lambda$   
 $(out1, out2).$   $S2\text{-}OT12$   $\sigma$   $out2)$   
 $\langle proof \rangle$

**definition**  $protocol\text{-}14\text{-}OT :: input1 \Rightarrow input2 \Rightarrow (unit \times bool)$  *spmf*  
**where**  $protocol\text{-}14\text{-}OT$   $M$   $C = do$  {  
 $let$   $(c0,c1) = C$ ;  
 $let$   $(m00, m01, m10, m11) = M$ ;  
 $S0 \leftarrow coin\text{-}spmf$ ;  
 $S1 \leftarrow coin\text{-}spmf$ ;  
 $S2 \leftarrow coin\text{-}spmf$ ;  
 $S3 \leftarrow coin\text{-}spmf$ ;

```

S4 ← coin-spmf;
S5 ← coin-spmf;
let a0 = S0 ⊕ S2 ⊕ m00;
let a1 = S0 ⊕ S3 ⊕ m01;
let a2 = S1 ⊕ S4 ⊕ m10;
let a3 = S1 ⊕ S5 ⊕ m11;
(·, Si) ← protocol-OT12 (S0, S1) c0;
(·, Sj) ← protocol-OT12 (S2, S3) c1;
(·, Sk) ← protocol-OT12 (S4, S5) c1;
let s2 = Si ⊕ (if c0 then Sk else Sj) ⊕ (if c0 then (if c1 then a3 else a2) else
(if c1 then a1 else a0));
return-spmf ((·), s2)}

```

**lemma** *lossless-protocol-14-OT*: *lossless-spmf (protocol-14-OT M C)*  
 ⟨*proof*⟩

**definition** *R1-14* :: *input1* ⇒ *input2* ⇒ '*v-OT121 view1* spmf  
**where** *R1-14* *msgs* *choice* = do {  
 let (*m*<sub>00</sub>, *m*<sub>01</sub>, *m*<sub>10</sub>, *m*<sub>11</sub>) = *msgs*;  
 let (*c*<sub>0</sub>, *c*<sub>1</sub>) = *choice*;  
*S*<sub>0</sub> :: bool ← coin-spmf;  
*S*<sub>1</sub> :: bool ← coin-spmf;  
*S*<sub>2</sub> :: bool ← coin-spmf;  
*S*<sub>3</sub> :: bool ← coin-spmf;  
*S*<sub>4</sub> :: bool ← coin-spmf;  
*S*<sub>5</sub> :: bool ← coin-spmf;  
*a* :: '*v-OT121* ← *R1-OT12* (*S*<sub>0</sub>, *S*<sub>1</sub>) *c*<sub>0</sub>;  
*b* :: '*v-OT121* ← *R1-OT12* (*S*<sub>2</sub>, *S*<sub>3</sub>) *c*<sub>1</sub>;  
*c* :: '*v-OT121* ← *R1-OT12* (*S*<sub>4</sub>, *S*<sub>5</sub>) *c*<sub>1</sub>;  
return-spmf (*msgs*, (*S*<sub>0</sub>, *S*<sub>1</sub>, *S*<sub>2</sub>, *S*<sub>3</sub>, *S*<sub>4</sub>, *S*<sub>5</sub>), *a*, *b*, *c*)}

**lemma** *lossless-R1-14*: *lossless-spmf (R1-14 msgs C)*  
 ⟨*proof*⟩

**definition** *R1-14-interm1* :: *input1* ⇒ *input2* ⇒ '*v-OT121 view1* spmf  
**where** *R1-14-interm1* *msgs* *choice* = do {  
 let (*m*<sub>00</sub>, *m*<sub>01</sub>, *m*<sub>10</sub>, *m*<sub>11</sub>) = *msgs*;  
 let (*c*<sub>0</sub>, *c*<sub>1</sub>) = *choice*;  
*S*<sub>0</sub> :: bool ← coin-spmf;  
*S*<sub>1</sub> :: bool ← coin-spmf;  
*S*<sub>2</sub> :: bool ← coin-spmf;  
*S*<sub>3</sub> :: bool ← coin-spmf;  
*S*<sub>4</sub> :: bool ← coin-spmf;  
*S*<sub>5</sub> :: bool ← coin-spmf;  
*a* :: '*v-OT121* ← *S1-OT12* (*S*<sub>0</sub>, *S*<sub>1</sub>) ();  
*b* :: '*v-OT121* ← *R1-OT12* (*S*<sub>2</sub>, *S*<sub>3</sub>) *c*<sub>1</sub>;  
*c* :: '*v-OT121* ← *R1-OT12* (*S*<sub>4</sub>, *S*<sub>5</sub>) *c*<sub>1</sub>;  
return-spmf (*msgs*, (*S*<sub>0</sub>, *S*<sub>1</sub>, *S*<sub>2</sub>, *S*<sub>3</sub>, *S*<sub>4</sub>, *S*<sub>5</sub>), *a*, *b*, *c*)}

**lemma** *lossless-R1-14-interm1*: *lossless-spmf* (*R1-14-interm1* *msgs* *C*)  
 ⟨*proof*⟩

**definition** *R1-14-interm2* :: *input1* ⇒ *input2* ⇒ '*v-OT121* *view1* *spmf*  
 where *R1-14-interm2* *msgs* *choice* = *do* {  
   *let* (*m00*, *m01*, *m10*, *m11*) = *msgs*;  
   *let* (*c0*, *c1*) = *choice*;  
   *S0* :: *bool* ← *coin-spmf*;  
   *S1* :: *bool* ← *coin-spmf*;  
   *S2* :: *bool* ← *coin-spmf*;  
   *S3* :: *bool* ← *coin-spmf*;  
   *S4* :: *bool* ← *coin-spmf*;  
   *S5* :: *bool* ← *coin-spmf*;  
   *a* :: '*v-OT121* ← *S1-OT12* (*S0*, *S1*) ();  
   *b* :: '*v-OT121* ← *S1-OT12* (*S2*, *S3*) ();  
   *c* :: '*v-OT121* ← *R1-OT12* (*S4*, *S5*) *c1*;  
   *return-spmf* (*msgs*, (*S0*, *S1*, *S2*, *S3*, *S4*, *S5*), *a*, *b*, *c*)}

**lemma** *lossless-R1-14-interm2*: *lossless-spmf* (*R1-14-interm2* *msgs* *C*)  
 ⟨*proof*⟩

**definition** *S1-14* :: *input1* ⇒ *unit* ⇒ '*v-OT121* *view1* *spmf*  
 where *S1-14* *msgs* - = *do* {  
   *let* (*m00*, *m01*, *m10*, *m11*) = *msgs*;  
   *S0* :: *bool* ← *coin-spmf*;  
   *S1* :: *bool* ← *coin-spmf*;  
   *S2* :: *bool* ← *coin-spmf*;  
   *S3* :: *bool* ← *coin-spmf*;  
   *S4* :: *bool* ← *coin-spmf*;  
   *S5* :: *bool* ← *coin-spmf*;  
   *a* :: '*v-OT121* ← *S1-OT12* (*S0*, *S1*) ();  
   *b* :: '*v-OT121* ← *S1-OT12* (*S2*, *S3*) ();  
   *c* :: '*v-OT121* ← *S1-OT12* (*S4*, *S5*) ();  
   *return-spmf* (*msgs*, (*S0*, *S1*, *S2*, *S3*, *S4*, *S5*), *a*, *b*, *c*)}

**lemma** *lossless-S1-14*: *lossless-spmf* (*S1-14* *m* *out*)  
 ⟨*proof*⟩

**lemma** *reduction-step1*:

**shows** ∃ *A1*. |*spmf* (*bind-spmf* (*R1-14* *M* (*c0*, *c1*)) *D*) *True* – *spmf* (*bind-spmf*  
 (*R1-14-interm1* *M* (*c0*, *c1*)) *D*) *True*| =  
 |*spmf* (*bind-spmf* (*pair-spmf* *coin-spmf* *coin-spmf*) (λ(*m0*, *m1*). *bind-spmf*  
 (*R1-OT12* (*m0*, *m1*) *c0*) (λ *view*. (*A1* *view* (*m0*, *m1*)))))) *True* –  
   *spmf* (*bind-spmf* (*pair-spmf* *coin-spmf* *coin-spmf*) (λ(*m0*, *m1*).  
*bind-spmf* (*S1-OT12* (*m0*, *m1*) ()) (λ *view*. (*A1* *view* (*m0*, *m1*)))))) *True*|

**including** *monad-normalisation*  
 ⟨*proof*⟩

**lemma** *reduction-step1'*:

**shows**  $| \text{spmf} (\text{bind-spmf} (\text{pair-spmf} \text{ coin-spmf} \text{ coin-spmf}) (\lambda(m0, m1). \text{bind-spmf} (\text{R1-OT12} (m0, m1) c0) (\lambda \text{view.} (A1 \text{ view} (m0, m1)))))) \text{ True} -$   
 $\text{spmf} (\text{bind-spmf} (\text{pair-spmf} \text{ coin-spmf} \text{ coin-spmf}) (\lambda(m0, m1). \text{bind-spmf} (\text{S1-OT12} (m0, m1) ()) (\lambda \text{view.} (A1 \text{ view} (m0, m1)))))) \text{ True} |$   
 $\leq \text{adv-OT12}$   
**(is ?lhs  $\leq$  adv-OT12)**  
 $\langle \text{proof} \rangle$

**lemma reduction-step2:**

**shows**  $\exists A1. | \text{spmf} (\text{bind-spmf} (\text{R1-14-interm1} M (c0, c1)) D) \text{ True} - \text{spmf} (\text{bind-spmf} (\text{R1-14-interm2} M (c0, c1)) D) \text{ True} | =$   
 $| \text{spmf} (\text{bind-spmf} (\text{pair-spmf} \text{ coin-spmf} \text{ coin-spmf}) (\lambda(m0, m1). \text{bind-spmf} (\text{R1-OT12} (m0, m1) c1) (\lambda \text{view.} (A1 \text{ view} (m0, m1)))))) \text{ True} -$   
 $\text{spmf} (\text{bind-spmf} (\text{pair-spmf} \text{ coin-spmf} \text{ coin-spmf}) (\lambda(m0, m1). \text{bind-spmf} (\text{S1-OT12} (m0, m1) ()) (\lambda \text{view.} (A1 \text{ view} (m0, m1)))))) \text{ True} |$   
 $\langle \text{proof} \rangle$   
**including monad-normalisation  $\langle \text{proof} \rangle$**

**lemma reduction-step3:**

**shows**  $\exists A1. | \text{spmf} (\text{bind-spmf} (\text{R1-14-interm2} M (c0, c1)) D) \text{ True} - \text{spmf} (\text{bind-spmf} (\text{S1-14} M \text{ out}) D) \text{ True} | =$   
 $| \text{spmf} (\text{bind-spmf} (\text{pair-spmf} \text{ coin-spmf} \text{ coin-spmf}) (\lambda(m0, m1). \text{bind-spmf} (\text{R1-OT12} (m0, m1) c1) (\lambda \text{view.} (A1 \text{ view} (m0, m1)))))) \text{ True} -$   
 $\text{spmf} (\text{bind-spmf} (\text{pair-spmf} \text{ coin-spmf} \text{ coin-spmf}) (\lambda(m0, m1). \text{bind-spmf} (\text{S1-OT12} (m0, m1) ()) (\lambda \text{view.} (A1 \text{ view} (m0, m1)))))) \text{ True} |$   
 $\langle \text{proof} \rangle$   
**including monad-normalisation  $\langle \text{proof} \rangle$**   
**including monad-normalisation  $\langle \text{proof} \rangle$**

**lemma reduction-P1-interm:**

**shows**  $| \text{spmf} (\text{bind-spmf} (\text{R1-14} M (c0, c1)) (D)) \text{ True} - \text{spmf} (\text{bind-spmf} (\text{S1-14} M \text{ out}) (D)) \text{ True} | \leq 3 * \text{adv-OT12}$   
**(is ?lhs  $\leq$  ?rhs)**  
 $\langle \text{proof} \rangle$

**lemma reduction-P1:**  $| \text{spmf} (\text{bind-spmf} (\text{R1-14} M (c0, c1)) (D)) \text{ True} - \text{spmf} (\text{funct-OT-14} M (c0, c1) \gg (\lambda (\text{out1}, \text{out2}). \text{S1-14} M \text{ out1} \gg (\lambda \text{view.} D \text{ view}))) \text{ True} |$   
 $\leq 3 * \text{adv-OT12}$   
 $\langle \text{proof} \rangle$

Party 2 security.

**lemma coin-coin:**  $\text{map-spmf} (\lambda S0. S0 \oplus S3 \oplus m1) \text{ coin-spmf} = \text{coin-spmf}$   
**(is ?lhs = ?rhs)**  
 $\langle \text{proof} \rangle$

**lemma coin-coin':**  $\text{map-spmf} (\lambda S3. S0 \oplus S3 \oplus m1) \text{ coin-spmf} = \text{coin-spmf}$   
 $\langle \text{proof} \rangle$

**definition**  $R2-14 :: input1 \Rightarrow input2 \Rightarrow 'v-OT122 \text{ view2 } spmf$   
**where**  $R2-14 \ M \ C = do \{$   
 $let \ (m0,m1,m2,m3) = M;$   
 $let \ (c0,c1) = C;$   
 $S0 :: bool \leftarrow coin-spmf;$   
 $S1 :: bool \leftarrow coin-spmf;$   
 $S2 :: bool \leftarrow coin-spmf;$   
 $S3 :: bool \leftarrow coin-spmf;$   
 $S4 :: bool \leftarrow coin-spmf;$   
 $S5 :: bool \leftarrow coin-spmf;$   
 $let \ a0 = S0 \oplus S2 \oplus m0;$   
 $let \ a1 = S0 \oplus S3 \oplus m1;$   
 $let \ a2 = S1 \oplus S4 \oplus m2;$   
 $let \ a3 = S1 \oplus S5 \oplus m3;$   
 $a :: 'v-OT122 \leftarrow R2-OT12 \ (S0,S1) \ c0;$   
 $b :: 'v-OT122 \leftarrow R2-OT12 \ (S2,S3) \ c1;$   
 $c :: 'v-OT122 \leftarrow R2-OT12 \ (S4,S5) \ c1;$   
 $return-spmf \ (C, (a0,a1,a2,a3), a,b,c)\}$

**lemma**  $lossless-R2-14: lossless-spmf \ (R2-14 \ M \ C)$   
 $\langle proof \rangle$

**definition**  $S2-14 :: input2 \Rightarrow bool \Rightarrow 'v-OT122 \text{ view2 } spmf$   
**where**  $S2-14 \ C \ out = do \{$   
 $let \ ((c0::bool),(c1::bool)) = C;$   
 $S0 :: bool \leftarrow coin-spmf;$   
 $S1 :: bool \leftarrow coin-spmf;$   
 $S2 :: bool \leftarrow coin-spmf;$   
 $S3 :: bool \leftarrow coin-spmf;$   
 $S4 :: bool \leftarrow coin-spmf;$   
 $S5 :: bool \leftarrow coin-spmf;$   
 $a0 :: bool \leftarrow coin-spmf;$   
 $a1 :: bool \leftarrow coin-spmf;$   
 $a2 :: bool \leftarrow coin-spmf;$   
 $a3 :: bool \leftarrow coin-spmf;$   
 $let \ a0' = (if \ ((\neg c0) \wedge (\neg c1)) \ then \ (S0 \oplus S2 \oplus out) \ else \ a0);$   
 $let \ a1' = (if \ ((\neg c0) \wedge c1) \ then \ (S0 \oplus S3 \oplus out) \ else \ a1);$   
 $let \ a2' = (if \ (c0 \wedge (\neg c1)) \ then \ (S1 \oplus S4 \oplus out) \ else \ a2);$   
 $let \ a3' = (if \ (c0 \wedge c1) \ then \ (S1 \oplus S5 \oplus out) \ else \ a3);$   
 $a :: 'v-OT122 \leftarrow S2-OT12 \ (c0::bool) \ (if \ c0 \ then \ S1 \ else \ S0);$   
 $b :: 'v-OT122 \leftarrow S2-OT12 \ (c1::bool) \ (if \ c1 \ then \ S3 \ else \ S2);$   
 $c :: 'v-OT122 \leftarrow S2-OT12 \ (c1::bool) \ (if \ c1 \ then \ S5 \ else \ S4);$   
 $return-spmf \ ((c0,c1), (a0',a1',a2',a3'), a,b,c)\}$

**lemma**  $lossless-S2-14: lossless-spmf \ (S2-14 \ c \ out)$   
 $\langle proof \rangle$

**lemma**  $P2-OT-14-FT: R2-14 \ (m0,m1,m2,m3) \ (False,True) = funct-OT-14 \ (m0,m1,m2,m3)$   
 $(False,True) \ggg \ (\lambda \ (out1, out2). S2-14 \ (False,True) \ out2)$



**including** *monad-normalisation*  
*<proof>*

**lemma** *P2-OT-14-TT*:  $R2-14 (m0, m1, m2, m3) (True, True) = \text{funct-OT-14} (m0, m1, m2, m3) (True, True) \gg (\lambda (out1, out2). S2-14 (True, True) out2)$   
**including** *monad-normalisation*  
*<proof>*

**lemma** *P2-OT-14-FF*:  $R2-14 (m0, m1, m2, m3) (False, False) = \text{funct-OT-14} (m0, m1, m2, m3) (False, False) \gg (\lambda (out1, out2). S2-14 (False, False) out2)$   
**including** *monad-normalisation*  
*<proof>*

**lemma** *P2-OT-14-TF*:  $R2-14 (m0, m1, m2, m3) (True, False) = \text{funct-OT-14} (m0, m1, m2, m3) (True, False) \gg (\lambda (out1, out2). S2-14 (True, False) out2)$   
**including** *monad-normalisation*  
*<proof>*

**lemma** *P2-sec-OT-14-split*:  $R2-14 (m0, m1, m2, m3) (c0, c1) = \text{funct-OT-14} (m0, m1, m2, m3) (c0, c1) \gg (\lambda (out1, out2). S2-14 (c0, c1) out2)$   
*<proof>*

**lemma** *P2-sec-OT-14*:  $R2-14 M C = \text{funct-OT-14} M C \gg (\lambda (out1, out2). S2-14 C out2)$   
*<proof>*

**sublocale** *OT-14*: *sim-det-def R1-14 S1-14 R2-14 S2-14 funct-OT-14 protocol-14-OT*  
*<proof>*

**lemma** *correctness-OT-14*:  
**shows**  $\text{funct-OT-14} M C = \text{protocol-14-OT} M C$   
*<proof>*

**lemma** *OT-14-correct*:  $OT-14.\text{correctness} M C$   
*<proof>*

**lemma** *OT-14-P2-sec*:  $OT-14.\text{perfect-sec-P2} m1 m2$   
*<proof>*

**lemma** *OT-14-P1-sec*:  $OT-14.\text{adv-P1} m1 m2 D \leq 3 * \text{adv-OT12}$   
*<proof>*

**end**

**locale** *OT-14-asymp* = *sim-det-def* +  
**fixes**  $S1-OT12 :: \text{nat} \Rightarrow (\text{bool} \times \text{bool}) \Rightarrow \text{unit} \Rightarrow 'v\text{-OT121} \text{ spmf}$   
**and**  $R1-OT12 :: \text{nat} \Rightarrow (\text{bool} \times \text{bool}) \Rightarrow \text{bool} \Rightarrow 'v\text{-OT121} \text{ spmf}$   
**and**  $\text{adv-OT12} :: \text{nat} \Rightarrow \text{real}$   
**and**  $S2-OT12 :: \text{nat} \Rightarrow \text{bool} \Rightarrow \text{bool} \Rightarrow 'v\text{-OT122} \text{ spmf}$

```

and  $R2\text{-}OT12 :: nat \Rightarrow (bool \times bool) \Rightarrow bool \Rightarrow 'v\text{-}OT122\text{ } spmf$ 
and  $protocol\text{-}OT12 :: (bool \times bool) \Rightarrow bool \Rightarrow (unit \times bool)\text{ } spmf$ 
assumes  $ot14\text{-}base: \bigwedge (n::nat). ot14\text{-}base (S1\text{-}OT12\text{ } n) (R1\text{-}12\text{-}0T\text{ } n) (adv\text{-}OT12\text{ } n) (S2\text{-}OT12\text{ } n) (R2\text{-}12OT\text{ } n) (protocol\text{-}OT12)$ 
begin

sublocale  $ot14\text{-}base (S1\text{-}OT12\text{ } n) (R1\text{-}12\text{-}0T\text{ } n) (adv\text{-}OT12\text{ } n) (S2\text{-}OT12\text{ } n) (R2\text{-}12OT\text{ } n) \langle proof \rangle$ 

lemma  $OT\text{-}14\text{-}P1\text{-}sec: OT\text{-}14.adv\text{-}P1 (R1\text{-}12\text{-}0T\text{ } n) n m1 m2 D \leq 3 * (adv\text{-}OT12\text{ } n)$ 
   $\langle proof \rangle$ 

theorem  $OT\text{-}14\text{-}P1\text{-}asym\text{-}sec: negligible (\lambda n. OT\text{-}14.adv\text{-}P1 (R1\text{-}12\text{-}0T\text{ } n) n m1 m2 D)$ 
if  $negligible (\lambda n. adv\text{-}OT12\text{ } n)$ 
   $\langle proof \rangle$ 

theorem  $OT\text{-}14\text{-}P2\text{-}asym\text{-}sec: OT\text{-}14.perfect\text{-}sec\text{-}P2 R2\text{-}OT12\text{ } n m1 m2$ 
   $\langle proof \rangle$ 

end

end

```

## 2.7 1-out-of-4 OT to GMW

We prove security for the gates of the GMW protocol in the semi honest model. We assume security on 1-out-of-4 OT.

**theory** *GMW* **imports**

*OT14*

**begin**

**type-synonym**  $share\text{-}1 = bool$

**type-synonym**  $share\text{-}2 = bool$

**type-synonym**  $shares\text{-}1 = bool\text{ } list$

**type-synonym**  $shares\text{-}2 = bool\text{ } list$

**type-synonym**  $msgs\text{-}14\text{-}OT = (bool \times bool \times bool \times bool)$

**type-synonym**  $choice\text{-}14\text{-}OT = (bool \times bool)$

**type-synonym**  $share\text{-}wire = (share\text{-}1 \times share\text{-}2)$

**locale**  $gmw\text{-}base =$

**fixes**  $S1\text{-}14\text{-}OT :: msgs\text{-}14\text{-}OT \Rightarrow unit \Rightarrow 'v\text{-}14\text{-}OT1\text{ } spmf$  — simulated view for party 1 of OT14

**and**  $R1\text{-}14\text{-}OT :: msgs\text{-}14\text{-}OT \Rightarrow choice\text{-}14\text{-}OT \Rightarrow 'v\text{-}14\text{-}OT1\text{ } spmf$  — real view for party 1 of OT14

**and**  $S2\text{-}14\text{-}OT :: choice\text{-}14\text{-}OT \Rightarrow bool \Rightarrow 'v\text{-}14\text{-}OT2\text{ } spmf$

**and**  $R2-14-OT :: \text{msgs-14-OT} \Rightarrow \text{choice-14-OT} \Rightarrow 'v-14-OT2 \text{ spmf}$   
**and**  $\text{protocol-14-OT} :: \text{msgs-14-OT} \Rightarrow \text{choice-14-OT} \Rightarrow (\text{unit} \times \text{bool}) \text{ spmf}$   
**and**  $\text{adv-14-OT} :: \text{real}$   
**assumes**  $P1-OT-14\text{-adv-bound}$ :  $\text{sim-det-def.adv-P1 } R1-14-OT \ S1-14-OT \ \text{funct-14-OT}$   
 $M \ C \ D \leq \text{adv-14-OT}$  — bound the advantage of party 1 in the 1-out-of-4 OT  
**and**  $P2-OT-12\text{-inf-theoretic}$ :  $\text{sim-det-def.perfect-sec-P2 } R2-14-OT \ S2-14-OT$   
 $\text{funct-14-OT } M \ C$  — information theoretic security for party 2 in the 1-out-of-4 OT  
**and**  $\text{correct-14}$ :  $\text{funct-OT-14 } \text{msgs } C = \text{protocol-14-OT } \text{msgs } C$  — correctness of the 1-out-of-4 OT  
**and**  $\text{lossless-R1-14-OT}$ :  $\text{lossless-spmf } (R1-14-OT \ (m1, m2, m3, m4) \ (c0, c1))$   
**and**  $\text{lossless-R2-14-OT}$ :  $\text{lossless-spmf } (R2-14-OT \ (m1, m2, m3, m4) \ (c0, c1))$   
**and**  $\text{lossless-S1-14-OT}$ :  $\text{lossless-spmf } (S1-14-OT \ (m1, m2, m3, m4) \ ())$   
**and**  $\text{lossless-S2-14-OT}$ :  $\text{lossless-spmf } (S2-14-OT \ (c0, c1) \ b)$   
**and**  $\text{lossless-protocol-14-OT}$ :  $\text{lossless-spmf } (\text{protocol-14-OT } S \ C)$   
**and**  $\text{lossless-funct-14-OT}$ :  $\text{lossless-spmf } (\text{funct-14-OT } M \ C)$   
**begin**

**lemma**  $\text{funct-14}$ :  $\text{funct-OT-14 } (m00, m01, m10, m11) \ (c0, c1)$   
 $= \text{return-spmf } ((), \text{if } c0 \text{ then } (\text{if } c1 \text{ then } m11 \text{ else } m10) \text{ else } (\text{if } c1 \text{ then } m01 \text{ else } m00))$   
 $\langle \text{proof} \rangle$

**sublocale**  $OT-14\text{-sim}$ :  $\text{sim-det-def } R1-14-OT \ S1-14-OT \ R2-14-OT \ S2-14-OT \ \text{funct-14-OT}$   
 $\text{protocol-14-OT}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{inf-th-14-OT-P4}$ :  $R2-14-OT \ \text{msgs } C = (\text{funct-OT-14 } \text{msgs } C \ggg (\lambda \ (s1, s2). \ S2-14-OT \ C \ s2))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{ass-adv-14-OT}$ :  $|\text{spmf } (\text{bind-spmf } (S1-14-OT \ \text{msgs } ()) \ (\lambda \ \text{view}. \ (D \ \text{view})))$   
 $\text{True} - \text{spmf } (\text{bind-spmf } (R1-14-OT \ \text{msgs } (c0, c1)) \ (\lambda \ \text{view}. \ (D \ \text{view})))$   
 $\text{True} \leq \text{adv-14-OT}$   
 $(\text{is } ?\text{lhs} \leq \text{adv-14-OT})$   
 $\langle \text{proof} \rangle$

The sharing scheme

**definition**  $\text{share} :: \text{bool} \Rightarrow \text{share-wire } \text{spmf}$   
**where**  $\text{share } x = \text{do } \{$   
 $\ a_1 \leftarrow \text{coin-spmf};$   
 $\ \text{let } b_1 = x \oplus a_1;$   
 $\ \text{return-spmf } (a_1, b_1)\}$

**lemma**  $\text{lossless-share } [\text{simp}]$ :  $\text{lossless-spmf } (\text{share } x)$   
 $\langle \text{proof} \rangle$

**definition**  $\text{reconstruct} :: (\text{share-1} \times \text{share-2}) \Rightarrow \text{bool } \text{spmf}$

**where** *reconstruct shares* = do {  
 let (a,b) = shares;  
 return-spmf (a  $\oplus$  b)}

**lemma** *lossless-reconstruct* [simp]: *lossless-spmf* (*reconstruct s*)  
 ⟨proof⟩

**lemma** *reconstruct-share* : (*bind-spmf* (*share x reconstruct*) = (*return-spmf x*)  
 ⟨proof⟩

**lemma** (*reconstruct (s1,s2)*  $\gg=$  ( $\lambda$  rec. *share rec*  $\gg=$  ( $\lambda$  shares. *reconstruct shares*)))  
 = *return-spmf (s1  $\oplus$  s2)*  
 ⟨proof⟩

**definition** *xor-evaluate* :: *bool*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool* *spmf*  
**where** *xor-evaluate A B* = *return-spmf (A  $\oplus$  B)*

**definition** *xor-funct* :: *share-wire*  $\Rightarrow$  *share-wire*  $\Rightarrow$  (*bool*  $\times$  *bool*) *spmf*  
**where** *xor-funct A B* = do {  
 let (a1, b1) = A;  
 let (a2,b2) = B;  
 return-spmf (a1  $\oplus$  a2, b1  $\oplus$  b2)}

**lemma** *lossless-xor-funct*: *lossless-spmf (xor-funct A B)*  
 ⟨proof⟩

**definition** *xor-protocol* :: *share-wire*  $\Rightarrow$  *share-wire*  $\Rightarrow$  (*bool*  $\times$  *bool*) *spmf*  
**where** *xor-protocol A B* = do {  
 let (a1, b1) = A;  
 let (a2,b2) = B;  
 return-spmf (a1  $\oplus$  a2, b1  $\oplus$  b2)}

**lemma** *lossless-xor-protocol*: *lossless-spmf (xor-protocol A B)*  
 ⟨proof⟩

**lemma** *share-xor-reconstruct*:  
**shows** *share x*  $\gg=$  ( $\lambda$  w1. *share y*  $\gg=$  ( $\lambda$  w2. *xor-protocol w1 w2*  
 $\gg=$  ( $\lambda$  (a, b). *reconstruct (a, b)*))) = *xor-evaluate x y*  
 ⟨proof⟩

**definition** *R1-xor* :: (*bool*  $\times$  *bool*)  $\Rightarrow$  (*bool*  $\times$  *bool*)  $\Rightarrow$  (*bool*  $\times$  *bool*) *spmf*  
**where** *R1-xor A B* = *return-spmf A*

**lemma** *lossless-R1-xor*: *lossless-spmf (R1-xor A B)*  
 ⟨proof⟩

**definition** *S1-xor* :: (*bool*  $\times$  *bool*)  $\Rightarrow$  *bool*  $\Rightarrow$  (*bool*  $\times$  *bool*) *spmf*  
**where** *S1-xor A out* = *return-spmf A*

**lemma** *lossless-S1-xor*: *lossless-spmf* (*S1-xor* *A* *out*)  
⟨*proof*⟩

**lemma** *P1-xor-inf-th*:  $R1\text{-xor } A \ B = \text{xor-funct } A \ B \ggg (\lambda (out1, out2). S1\text{-xor } A \ out1)$   
⟨*proof*⟩

**definition** *R2-xor* ::  $(bool \times bool) \Rightarrow (bool \times bool) \Rightarrow (bool \times bool) \text{ spmf}$   
**where** *R2-xor* *A* *B* = *return-spmf* *B*

**lemma** *lossless-R2-xor*: *lossless-spmf* (*R2-xor* *A* *B*)  
⟨*proof*⟩

**definition** *S2-xor* ::  $(bool \times bool) \Rightarrow bool \Rightarrow (bool \times bool) \text{ spmf}$   
**where** *S2-xor* *B* *out* = *return-spmf* *B*

**lemma** *lossless-S2-xor*: *lossless-spmf* (*S2-xor* *A* *out*)  
⟨*proof*⟩

**lemma** *P2-xor-inf-th*:  $R2\text{-xor } A \ B = \text{xor-funct } A \ B \ggg (\lambda (out1, out2). S2\text{-xor } B \ out2)$   
⟨*proof*⟩

**sublocale** *xor-sim-det*: *sim-det-def* *R1-xor* *S1-xor* *R2-xor* *S2-xor* *xor-funct* *xor-protocol*  
⟨*proof*⟩

**lemma** *xor-sim-det.perfect-sec-P1* *m1* *m2*  
⟨*proof*⟩

**lemma** *xor-sim-det.perfect-sec-P2* *m1* *m2*  
⟨*proof*⟩

**definition** *and-funct* ::  $(share\text{-}1 \times share\text{-}2) \Rightarrow (share\text{-}1 \times share\text{-}2) \Rightarrow share\text{-}wire \text{ spmf}$   
**where** *and-funct* *A* *B* = *do* {  
  *let* (*a1*, *a2*) = *A*;  
  *let* (*b1*, *b2*) = *B*;  
   $\sigma \leftarrow coin\text{-}spmf$ ;  
  *return-spmf* ( $\sigma, \sigma \oplus ((a1 \oplus b1) \wedge (a2 \oplus b2))$ )}

**lemma** *lossless-and-funct*: *lossless-spmf* (*and-funct* *A* *B*)  
⟨*proof*⟩

**definition** *and-evaluate* ::  $bool \Rightarrow bool \Rightarrow bool \text{ spmf}$   
**where** *and-evaluate* *A* *B* = *return-spmf* (*A*  $\wedge$  *B*)

**definition** *and-protocol* ::  $share\text{-}wire \Rightarrow share\text{-}wire \Rightarrow share\text{-}wire \text{ spmf}$   
**where** *and-protocol* *A* *B* = *do* {

```

let (a1, b1) = A;
let (a2, b2) = B;
σ ← coin-spmf;
let s0 = σ ⊕ ((a1 ⊕ False) ∧ (b1 ⊕ False));
let s1 = σ ⊕ ((a1 ⊕ False) ∧ (b1 ⊕ True));
let s2 = σ ⊕ ((a1 ⊕ True) ∧ (b1 ⊕ False));
let s3 = σ ⊕ ((a1 ⊕ True) ∧ (b1 ⊕ True));
(·, s) ← protocol-14-OT (s0, s1, s2, s3) (a2, b2);
return-spmf (σ, s)

```

**lemma** *lossless-and-protocol*: *lossless-spmf* (*and-protocol* A B)  
 ⟨*proof*⟩

**lemma** *and-correct*: *and-protocol* (a1, b1) (a2, b2) = *and-funct* (a1, b1) (a2, b2)  
 ⟨*proof*⟩

**lemma** *share-and-reconstruct*:

**shows** *share* x ≫≧ (λ (a1, a2). *share* y ≫≧ (λ (b1, b2).  
*and-protocol* (a1, b1) (a2, b2) ≫≧ (λ (a, b). *reconstruct* (a, b)))) =  
*and-evaluate* x y  
 ⟨*proof*⟩

**definition** *and-R1* :: (*share-1* × *share-1*) ⇒ (*share-2* × *share-2*) ⇒ (((*share-1* ×  
*share-1*) × *bool* × 'v-14-OT1) × (*share-1* × *share-2*)) *spmf*

**where** *and-R1* A B = *do* {  
 let (a1, a2) = A;  
 let (b1, b2) = B;  
 σ ← coin-spmf;  
 let s0 = σ ⊕ ((a1 ⊕ False) ∧ (a2 ⊕ False));  
 let s1 = σ ⊕ ((a1 ⊕ False) ∧ (a2 ⊕ True));  
 let s2 = σ ⊕ ((a1 ⊕ True) ∧ (a2 ⊕ False));  
 let s3 = σ ⊕ ((a1 ⊕ True) ∧ (a2 ⊕ True));  
 V ← R1-14-OT (s0, s1, s2, s3) (b1, b2);  
 (·, s) ← protocol-14-OT (s0, s1, s2, s3) (b1, b2);  
 return-spmf (((a1, a2), σ, V), (σ, s))
}

**lemma** *lossless-and-R1*: *lossless-spmf* (*and-R1* A B)  
 ⟨*proof*⟩

**definition** *S1-and* :: (*share-1* × *share-1*) ⇒ *bool* ⇒ (((*bool* × *bool*) × *bool* ×  
 'v-14-OT1)) *spmf*

**where** *S1-and* A σ = *do* {  
 let (a1, a2) = A;  
 let s0 = σ ⊕ ((a1 ⊕ False) ∧ (a2 ⊕ False));  
 let s1 = σ ⊕ ((a1 ⊕ False) ∧ (a2 ⊕ True));  
 let s2 = σ ⊕ ((a1 ⊕ True) ∧ (a2 ⊕ False));  
 let s3 = σ ⊕ ((a1 ⊕ True) ∧ (a2 ⊕ True));  
 V ← S1-14-OT (s0, s1, s2, s3) ();  
 return-spmf ((a1, a2), σ, V)
}

**definition** *out1* :: (*share-1* × *share-1*) ⇒ (*share-2* × *share-2*) ⇒ *bool* ⇒ (*share-1* × *share-2*) *spmf*  
**where** *out1* *A B*  $\sigma = do$  {  
  *let* (*a1,a2*) = *A*;  
  *let* (*b1,b2*) = *B*;  
  *return-spmf* ( $\sigma, \sigma \oplus ((a1 \oplus b1) \wedge (a2 \oplus b2))$ )}

**definition** *S1-and'* :: (*share-1* × *share-1*) ⇒ (*share-2* × *share-2*) ⇒ *bool* ⇒ (((*bool* × *bool*) × *bool* × '*v-14-OT1*) × (*share-1* × *share-2*)) *spmf*  
**where** *S1-and'* *A B*  $\sigma = do$  {  
  *let* (*a1,a2*) = *A*;  
  *let* (*b1,b2*) = *B*;  
  *let* *s0* =  $\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus False))$ ;  
  *let* *s1* =  $\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus True))$ ;  
  *let* *s2* =  $\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus False))$ ;  
  *let* *s3* =  $\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus True))$ ;  
  *V* ← *S1-14-OT* (*s0,s1,s2,s3*) ();  
  *return-spmf* (((*a1,a2*),  $\sigma, V$ ), ( $\sigma, \sigma \oplus ((a1 \oplus b1) \wedge (a2 \oplus b2))$ ))}

**lemma** *sec-ex-P1-and*:

**shows**  $\exists (A :: 'v-14-OT1 \Rightarrow bool \Rightarrow bool \text{ spmf}).$   
  |*spmf* ((*and-funct* (*a1, a2*) (*b1,b2*))  $\ggg (\lambda (s1, s2). (S1\text{-and}' (a1,a2) (b1,b2) s1)$   
 $\ggg (D :: (((bool \times bool) \times bool \times 'v-14-OT1) \times (share-1 \times share-2)) \Rightarrow bool \text{ spmf})) True - \text{spmf} ((and-R1 (a1, a2) (b1,b2)) \ggg D) True| =$   
  |*spmf* (*coin-spmf*  $\ggg (\lambda \sigma. S1\text{-14-OT} ((\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus True))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus True))))$ )  
 $\ggg (\lambda \text{view}. A \text{ view } \sigma)) True$   
  - *spmf* (*coin-spmf*  $\ggg (\lambda \sigma. R1\text{-14-OT} ((\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus True))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus True))))$ ) (*b1, b2*)  
 $\ggg (\lambda \text{view}. A \text{ view } \sigma)) True|$

**including** *monad-normalisation*

*<proof>*

**lemma** *bound-14-OT*:

|*spmf* (*coin-spmf*  $\ggg (\lambda \sigma. S1\text{-14-OT} ((\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus True))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus True))))$ )  
 $\ggg (\lambda \text{view}. (A :: 'v-14-OT1 \Rightarrow bool \Rightarrow bool \text{ spmf}) \text{ view } \sigma)) True - \text{spmf}$   
(*coin-spmf*  $\ggg (\lambda \sigma. R1\text{-14-OT} ((\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus False) \wedge (a2 \oplus True))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus False))), (\sigma \oplus ((a1 \oplus True) \wedge (a2 \oplus True))))$ ) (*b1, b2*)  
 $\ggg (\lambda \text{view}. A \text{ view } \sigma)) True| \leq \text{adv-14-OT}$

**(is ?lhs** ≤ *adv-14-OT*)

*<proof>*

**lemma** *security-and-P1*:

**shows**  $| \text{spmf } ((\text{and-funct } (a1, a2) (b1, b2)) \gg (\lambda (s1, s2). (S1\text{-and}' (a1, a2) (b1, b2) s1)) \gg (D :: (((\text{bool} \times \text{bool}) \times \text{bool} \times 'v\text{-}14\text{-OT}1) \times (\text{share-}1 \times \text{share-}2)) \Rightarrow \text{bool } \text{spmf}))) \text{ True} - \text{spmf } ((\text{and-R}1 (a1, a2) (b1, b2)) \gg D) \text{ True} | \leq \text{adv-}14\text{-OT}$   
 $\langle \text{proof} \rangle$

**lemma** *security-and-P1'*:

**shows**  $| \text{spmf } ((\text{and-R}1 (a1, a2) (b1, b2)) \gg D) \text{ True} - \text{spmf } ((\text{and-funct } (a1, a2) (b1, b2)) \gg (\lambda (s1, s2). (S1\text{-and}' (a1, a2) (b1, b2) s1)) \gg (D :: (((\text{bool} \times \text{bool}) \times \text{bool} \times 'v\text{-}14\text{-OT}1) \times (\text{share-}1 \times \text{share-}2)) \Rightarrow \text{bool } \text{spmf}))) \text{ True} | \leq \text{adv-}14\text{-OT}$   
 $\langle \text{proof} \rangle$

**definition** *and-R2* ::  $(\text{share-}1 \times \text{share-}2) \Rightarrow (\text{share-}2 \times \text{share-}1) \Rightarrow (((\text{bool} \times \text{bool}) \times 'v\text{-}14\text{-OT}2) \times (\text{share-}1 \times \text{share-}2)) \text{ spmf}$

**where** *and-R2*  $A B = \text{do } \{$   
 $\text{let } (a1, a2) = A;$   
 $\text{let } (b1, b2) = B;$   
 $\sigma \leftarrow \text{coin-spmf};$   
 $\text{let } s0 = \sigma \oplus ((a1 \oplus \text{False}) \wedge (a2 \oplus \text{False}));$   
 $\text{let } s1 = \sigma \oplus ((a1 \oplus \text{False}) \wedge (a2 \oplus \text{True}));$   
 $\text{let } s2 = \sigma \oplus ((a1 \oplus \text{True}) \wedge (a2 \oplus \text{False}));$   
 $\text{let } s3 = \sigma \oplus ((a1 \oplus \text{True}) \wedge (a2 \oplus \text{True}));$   
 $(-, s) \leftarrow \text{protocol-}14\text{-OT } (s0, s1, s2, s3) B;$   
 $V \leftarrow \text{R}2\text{-}14\text{-OT } (s0, s1, s2, s3) B;$   
 $\text{return-spmf } ((B, V), (\sigma, s)) \}$

**lemma** *lossless-and-R2*:  $\text{lossless-spmf } (\text{and-R}2 A B)$

$\langle \text{proof} \rangle$

**definition** *S2-and* ::  $(\text{share-}1 \times \text{share-}2) \Rightarrow \text{bool} \Rightarrow (((\text{bool} \times \text{bool}) \times 'v\text{-}14\text{-OT}2)) \text{ spmf}$

**where** *S2-and*  $B s2 = \text{do } \{$   
 $\text{let } (a2, b2) = B;$   
 $V :: 'v\text{-}14\text{-OT}2 \leftarrow \text{S}2\text{-}14\text{-OT } (a2, b2) s2;$   
 $\text{return-spmf } ((B, V)) \}$

**definition** *out2* ::  $(\text{share-}1 \times \text{share-}2) \Rightarrow (\text{share-}1 \times \text{share-}2) \Rightarrow \text{bool} \Rightarrow (\text{share-}1 \times \text{share-}2) \text{ spmf}$

**where** *out2*  $B A s2 = \text{do } \{$   
 $\text{let } (a1, b1) = A;$   
 $\text{let } (a2, b2) = B;$   
 $\text{let } s1 = s2 \oplus ((a1 \oplus a2) \wedge (b1 \oplus b2));$   
 $\text{return-spmf } (s1, s2) \}$

**definition** *S2-and'* ::  $(\text{share-}1 \times \text{share-}2) \Rightarrow (\text{share-}1 \times \text{share-}2) \Rightarrow \text{bool} \Rightarrow (((\text{bool}$



$\times \text{bool}) \times 'v\text{-}14\text{-}OT2) \times (\text{share}\text{-}1 \times \text{share}\text{-}2)) \text{ spmf}$   
**where**  $S2\text{-and}' B A s2 = \text{do} \{$   
 $\text{let } (a1, a2) = A;$   
 $\text{let } (b1, b2) = B;$   
 $V :: 'v\text{-}14\text{-}OT2 \leftarrow S2\text{-}14\text{-}OT B s2;$   
 $\text{let } s1 = s2 \oplus ((a1 \oplus b1) \wedge (a2 \oplus b2));$   
 $\text{return}\text{-}\text{spmf } ((B, V), s1, s2)\}$

**lemma** *lossless-S2-and: lossless-spmf (S2-and B s2)*  
 $\langle \text{proof} \rangle$

**sublocale** *and-secret-sharing: sim-non-det-def and-R1 S1-and out1 and-R2 S2-and out2 and-funct*  $\langle \text{proof} \rangle$

**lemma** *ideal-S1-and: and-secret-sharing.Ideal1 (a1, b1) (a2, b2) s2 = S1-and' (a1, b1) (a2, b2) s2*  
 $\langle \text{proof} \rangle$

**lemma** *and-P2-security: and-secret-sharing.perfect-sec-P2 m1 m2*  
 $\langle \text{proof} \rangle$

**lemma** *and-P1-security: and-secret-sharing.adv-P1 m1 m2 D  $\leq$  adv-14-OT*  
 $\langle \text{proof} \rangle$

**definition**  $F = \{\text{and-evaluate}, \text{xor-evaluate}\}$

**lemma** *share-reconstruct-xor: share x  $\ggg$  ( $\lambda(a1, a2). \text{share } y \ggg (\lambda(b1, b2). \text{xor-protocol } (a1, b1) (a2, b2) \ggg (\lambda(a, b). \text{reconstruct } (a, b)))$ ) = xor-evaluate x y*  
 $\langle \text{proof} \rangle$

**sublocale** *share-correct: secret-sharing-scheme share reconstruct F*  $\langle \text{proof} \rangle$

**lemma** *share-correct.sharing-correct input*  
 $\langle \text{proof} \rangle$

**lemma** *share-correct.correct-share-eval input1 input2*  
 $\langle \text{proof} \rangle$

**end**

**locale** *gmw-asym =*

**fixes**  $S1\text{-}14\text{-}OT :: \text{nat} \Rightarrow \text{msgs}\text{-}14\text{-}OT \Rightarrow \text{unit} \Rightarrow 'v\text{-}14\text{-}OT1 \text{ spmf}$   
**and**  $R1\text{-}14\text{-}OT :: \text{nat} \Rightarrow \text{msgs}\text{-}14\text{-}OT \Rightarrow \text{choice}\text{-}14\text{-}OT \Rightarrow 'v\text{-}14\text{-}OT1 \text{ spmf}$   
**and**  $S2\text{-}14\text{-}OT :: \text{nat} \Rightarrow \text{choice}\text{-}14\text{-}OT \Rightarrow \text{bool} \Rightarrow 'v\text{-}14\text{-}OT2 \text{ spmf}$   
**and**  $R2\text{-}14\text{-}OT :: \text{nat} \Rightarrow \text{msgs}\text{-}14\text{-}OT \Rightarrow \text{choice}\text{-}14\text{-}OT \Rightarrow 'v\text{-}14\text{-}OT2 \text{ spmf}$   
**and**  $\text{protocol}\text{-}14\text{-}OT :: \text{nat} \Rightarrow \text{msgs}\text{-}14\text{-}OT \Rightarrow \text{choice}\text{-}14\text{-}OT \Rightarrow (\text{unit} \times \text{bool}) \text{ spmf}$   
**and**  $\text{adv}\text{-}14\text{-}OT :: \text{nat} \Rightarrow \text{real}$

```

assumes gmw-base:  $\bigwedge (n::nat). \text{gmw-base } (S1-14-OT\ n) (R1-14-OT\ n) (S2-14-OT\ n) (R2-14-OT\ n) (\text{protocol-14-OT}\ n) (\text{adv-14-OT}\ n)$ 
begin

sublocale gmw-base (S1-14-OT n) (R1-14-OT n) (S2-14-OT n) (R2-14-OT n)
(protocol-14-OT n) (adv-14-OT n)
  <proof>

lemma xor-sim-det.perfect-sec-P1 m1 m2
  <proof>

lemma xor-sim-det.perfect-sec-P2 m1 m2
  <proof>

lemma and-P1-adv-negligible:
  assumes negligible ( $\lambda n. \text{adv-14-OT}\ n$ )
  shows negligible ( $\lambda n. \text{and-secret-sharing.adv-P1}\ n\ m1\ m2\ D$ )
  <proof>

lemma and-P2-security: and-secret-sharing.perfect-sec-P2 n m1 m2
  <proof>

end

end

```

## 2.8 Secure multiplication protocol

```

theory Secure-Multiplication imports
  CryptHOL.Cyclic-Group-SPMF
  Uniform-Sampling
  Semi-Honest-Def
begin

locale secure-mult =
  fixes q :: nat
  assumes q-gt-0:  $q > 0$ 
  and prime q
begin

type-synonym real-view =  $nat \Rightarrow nat \Rightarrow ((nat \times nat \times nat \times nat) \times nat \times nat) \text{ spmf}$ 
type-synonym sim =  $nat \Rightarrow nat \Rightarrow ((nat \times nat \times nat \times nat) \times nat \times nat) \text{ spmf}$ 

lemma samp-uni-set-spmf [simp]:  $\text{set-spmf } (\text{sample-uniform}\ q) = \{..< q\}$ 
  <proof>

definition funct ::  $nat \Rightarrow nat \Rightarrow (nat \times nat) \text{ spmf}$ 

```

**where** *funct*  $x\ y = do \{$   
 $s \leftarrow sample\text{-}uniform\ q;$   
 $return\text{-}spmf\ (s, (x*y + (q - s))\ mod\ q)\}$

**definition**  $TI :: ((nat \times nat) \times (nat \times nat))\ spmf$   
**where**  $TI = do \{$   
 $a \leftarrow sample\text{-}uniform\ q;$   
 $b \leftarrow sample\text{-}uniform\ q;$   
 $r \leftarrow sample\text{-}uniform\ q;$   
 $return\text{-}spmf\ ((a, r), (b, ((a*b + (q - r))\ mod\ q)))\}$

**definition**  $out :: nat \Rightarrow nat \Rightarrow (nat \times nat)\ spmf$   
**where**  $out\ x\ y = do \{$   
 $((c1, d1), (c2, d2)) \leftarrow TI;$   
 $let\ e2 = (x + c1)\ mod\ q;$   
 $let\ e1 = (y + (q - c2))\ mod\ q;$   
 $return\text{-}spmf\ (((x*e1 + (q - d1))\ mod\ q), ((e2 * c2 + (q - d2))\ mod\ q))\}$

**definition**  $R1 :: real\text{-}view$   
**where**  $R1\ x\ y = do \{$   
 $((c1, d1), (c2, d2)) \leftarrow TI;$   
 $let\ e2 = (x + c1)\ mod\ q;$   
 $let\ e1 = (y + (q - c2))\ mod\ q;$   
 $let\ s1 = (x*e1 + (q - d1))\ mod\ q;$   
 $let\ s2 = (e2 * c2 + (q - d2))\ mod\ q;$   
 $return\text{-}spmf\ ((x, c1, d1, e1), s1, s2)\}$

**definition**  $S1 :: nat \Rightarrow nat \Rightarrow (nat \times nat \times nat \times nat)\ spmf$   
**where**  $S1\ x\ s1 = do \{$   
 $a :: nat \leftarrow sample\text{-}uniform\ q;$   
 $e1 \leftarrow sample\text{-}uniform\ q;$   
 $let\ d1 = (x*e1 + (q - s1))\ mod\ q;$   
 $return\text{-}spmf\ (x, a, d1, e1)\}$

**definition**  $Out1 :: nat \Rightarrow nat \Rightarrow nat \Rightarrow (nat \times nat)\ spmf$   
**where**  $Out1\ x\ y\ s1 = do \{$   
 $let\ s2 = (x*y + (q - s1))\ mod\ q;$   
 $return\text{-}spmf\ (s1, s2)\}$

**definition**  $R2 :: real\text{-}view$   
**where**  $R2\ x\ y = do \{$   
 $((c1, d1), (c2, d2)) \leftarrow TI;$   
 $let\ e2 = (x + c1)\ mod\ q;$   
 $let\ e1 = (y + (q - c2))\ mod\ q;$   
 $let\ s1 = (x*e1 + (q - d1))\ mod\ q;$   
 $let\ s2 = (e2 * c2 + (q - d2))\ mod\ q;$   
 $return\text{-}spmf\ ((y, c2, d2, e2), s1, s2)\}$

**definition**  $S2 :: nat \Rightarrow nat \Rightarrow (nat \times nat \times nat \times nat)\ spmf$

**where**  $S2\ y\ s2 = do \{$   
 $b \leftarrow sample\text{-}uniform\ q;$   
 $e2 \leftarrow sample\text{-}uniform\ q;$   
 $let\ d2 = (e2*b + (q - s2))\ mod\ q;$   
 $return\text{-}spmf\ (y, b, d2, e2)\}$

**definition**  $Out2 :: nat \Rightarrow nat \Rightarrow nat \Rightarrow (nat \times nat)\ spmf$

**where**  $Out2\ y\ x\ s2 = do \{$   
 $let\ s1 = (x*y + (q - s2))\ mod\ q;$   
 $return\text{-}spmf\ (s1, s2)\}$

**definition**  $Ideal2 :: nat \Rightarrow nat \Rightarrow nat \Rightarrow ((nat \times nat \times nat \times nat) \times (nat \times nat))\ spmf$

**where**  $Ideal2\ y\ x\ out2 = do \{$   
 $view2 :: (nat \times nat \times nat \times nat) \leftarrow S2\ y\ out2;$   
 $out2 \leftarrow Out2\ y\ x\ out2;$   
 $return\text{-}spmf\ (view2, out2)\}$

**sublocale**  $sim\text{-}non\text{-}det\text{-}def: sim\text{-}non\text{-}det\text{-}def\ R1\ S1\ Out1\ R2\ S2\ Out2\ funct\ \langle proof \rangle$

**lemma**  $minus\text{-}mod:$

**assumes**  $a > b$   
**shows**  $[a - b\ mod\ q = a - b]\ (mod\ q)$   
 $\langle proof \rangle$

**lemma**  $q\text{-}cong:[a = q + a]\ (mod\ q)$

$\langle proof \rangle$

**lemma**  $q\text{-}cong\text{-}reverse:[q + a = a]\ (mod\ q)$

$\langle proof \rangle$

**lemma**  $qq\text{-}cong:[a = q*q + a]\ (mod\ q)$

$\langle proof \rangle$

**lemma**  $minus\text{-}q\text{-}mult\text{-}cancel:$

**assumes**  $[a = e + b - q * c - d]\ (mod\ q)$

**and**  $e + b - d > 0$

**and**  $e + b - q * c - d > 0$

**shows**  $[a = e + b - d]\ (mod\ q)$

$\langle proof \rangle$

**lemma**  $s1\text{-}s2:$

**assumes**  $x < q\ a < q\ b < q$  **and**  $r:r < q\ y < q$

**shows**  $((x + a)\ mod\ q * b + q - (a * b + q - r)\ mod\ q)\ mod\ q =$

$(x * y + q - (x * ((y + q - b)\ mod\ q) + q - r)\ mod\ q)\ mod\ q$

$\langle proof \rangle$

**lemma**  $s1\text{-}s2\text{-}P2:$

**assumes**  $x < q\ xa < q\ xb < q\ xc < q\ y < q$

**shows**  $((y, xa, (xb * xa + q - xc) \bmod q, (x + xb) \bmod q), (x * ((y + q - xa) \bmod q) + q - xc) \bmod q, ((x + xb) \bmod q * xa + q - (xb * xa + q - xc) \bmod q) \bmod q) =$   
 $((y, xa, (xb * xa + q - xc) \bmod q, (x + xb) \bmod q), (x * ((y + q - xa) \bmod q) + q - xc) \bmod q, (x * y + q - (x * ((y + q - xa) \bmod q) + q - xc) \bmod q) \bmod q)$   
 $\langle proof \rangle$

**lemma c1:**

**assumes**  $e2 = (x + c1) \bmod q$   
**and**  $x < q$   $c1 < q$   
**shows**  $c1 = (e2 + q - x) \bmod q$   
 $\langle proof \rangle$

**lemma c1-P2:**

**assumes**  $xb < q$   $xa < q$   $xc < q$   $x < q$   
**shows**  $((y, xa, (xb * xa + q - xc) \bmod q, (x + xb) \bmod q), (x * ((y + q - xa) \bmod q) + q - xc) \bmod q, (x * y + q - (x * ((y + q - xa) \bmod q) + q - xc) \bmod q) \bmod q) =$   
 $((y, xa, (((x + xb) \bmod q + q - x) \bmod q * xa + q - xc) \bmod q, (x + xb) \bmod q), (x * ((y + q - xa) \bmod q) + q - xc) \bmod q, (x * y + q - (x * ((y + q - xa) \bmod q) + q - xc) \bmod q) \bmod q)$   
 $\langle proof \rangle$

**lemma minus-mod-cancel:**

**assumes**  $a - b > 0$   $a - b \bmod q > 0$   
**shows**  $[a - b + c = a - b \bmod q + c] \bmod q$   
 $\langle proof \rangle$

**lemma d2:**

**assumes**  $d2: d2 = (((e2 + q - x) \bmod q) * b + (q - r)) \bmod q$   
**and**  $s1: s1 = (x * ((y + (q - b)) \bmod q) + (q - r)) \bmod q$   
**and**  $s2: s2 = (x * y + (q - s1)) \bmod q$   
**and**  $x: x < q$   
**and**  $y: y < q$   
**and**  $r: r < q$   
**and**  $b: b < q$   
**and**  $e2: e2 < q$   
**shows**  $d2 = (e2 * b + (q - s2)) \bmod q$   
 $\langle proof \rangle$

**lemma d2-P2:**

**assumes**  $x: x < q$  **and**  $y: y < q$  **and**  $r: r < q$  **and**  $b: b < q$  **and**  $e2: e2 < q$   
**shows**  $((y, b, ((e2 + q - x) \bmod q * b + q - r) \bmod q, e2), (x * ((y + q - b) \bmod q) + q - r) \bmod q, (x * y + q - (x * ((y + q - b) \bmod q) + q - r) \bmod q) \bmod q) =$   
 $((y, b, (e2 * b + q - (x * y + q - (x * ((y + q - b) \bmod q) + q - r) \bmod q) \bmod q) \bmod q, e2), (x * ((y + q - b) \bmod q) + q - r) \bmod q,$   
 $(x * y + q - (x * ((y + q - b) \bmod q) + q - r) \bmod q) \bmod q)$

*<proof>*

**lemma** *s1*:

**assumes** *s2*:  $s2 = (x*y + q - s1) \bmod q$

**and** *x*:  $x < q$

**and** *y*:  $y < q$

**and** *s1*:  $s1 < q$

**shows**  $s1 = (x*y + q - s2) \bmod q$

*<proof>*

**lemma** *s1-P2*:

**assumes** *x*:  $x < q$

**and** *y*:  $y < q$

**and**  $b < q$

**and**  $e2 < q$

**and**  $r < q$

**and**  $s1 < q$

**shows**  $((y, b, (e2 * b + q - (x * y + q - r) \bmod q) \bmod q, e2), r, (x * y + q - r) \bmod q) =$

$((y, b, (e2 * b + q - (x * y + q - r) \bmod q) \bmod q, e2), (x * y + q - (x * y + q - r) \bmod q) \bmod q, (x * y + q - r) \bmod q)$

*<proof>*

**theorem** *P2-security*:

**assumes**  $x < q$   $y < q$

**shows** *sim-non-det-def.perfect-sec-P2* *x y*

**including** *monad-normalisation*

*<proof>*

**lemma** *s1-s2-P1*: **assumes**  $x < q$   $xa < q$   $xb < q$   $xc < q$   $y < q$

**shows**  $((x, xa, xb, (y + q - xc) \bmod q), (x * ((y + q - xc) \bmod q) + q - xb) \bmod q, ((x + xa) \bmod q * xc + q - (xa * xc + q - xb) \bmod q) \bmod q) =$

$((x, xa, xb, (y + q - xc) \bmod q), (x * ((y + q - xc) \bmod q) + q - xb) \bmod q, (x * y + q - (x * ((y + q - xc) \bmod q) + q - xb) \bmod q) \bmod q)$

*<proof>*

**lemma** *mod-minus*: **assumes**  $a - b > 0$  **and**  $c - d > 0$

**shows**  $(a - b + (c - d \bmod q)) \bmod q = (a - b + (c - d)) \bmod q$

*<proof>*

**lemma** *r*:

**assumes** *e1*:  $e1 = (y + (q - b)) \bmod q$

**and** *s1*:  $s1 = (x*((y + (q - b)) \bmod q) + (q - r)) \bmod q$

**and** *b*:  $b < q$

**and** *x*:  $x < q$

**and** *y*:  $y < q$

**and** *r*:  $r < q$

**shows**  $r = (x*e1 + (q - s1)) \bmod q$

(**is** *?lhs = ?rhs*)

*<proof>*

**lemma** *r-P2*:

**assumes** *b*:  $b < q$  **and** *x*:  $x < q$  **and** *y*:  $y < q$  **and** *r*:  $r < q$

**shows**

$((x, a, r, (y + q - b) \bmod q), (x * ((y + q - b) \bmod q) + q - r) \bmod q, (x * y + q - (x * ((y + q - b) \bmod q) + q - r) \bmod q) \bmod q) =$

$((x, a, (x * ((y + q - b) \bmod q) + q - (x * ((y + q - b) \bmod q) + q - r) \bmod q) \bmod q, (y + q - b) \bmod q, (x * ((y + q - b) \bmod q) + q - r) \bmod q,$

$(x * y + q - (x * ((y + q - b) \bmod q) + q - r) \bmod q) \bmod q)$

*<proof>*

**theorem** *P1-security*:

**assumes**  $x < q$   $y < q$

**shows** *sim-non-det-def.perfect-sec-P1*  $x$   $y$

**including** *monad-normalisation*

*<proof>*

**end**

**locale** *secure-mult-asymp* =

**fixes**  $q :: \text{nat} \Rightarrow \text{nat}$

**assumes**  $\bigwedge n. \text{secure-mult } (q \ n)$

**begin**

**sublocale** *secure-mult*  $q$   $n$  **for**  $n$

*<proof>*

**theorem** *P1-secure*:

**assumes**  $x < q$   $n$   $y < q$   $n$

**shows** *sim-non-det-def.perfect-sec-P1*  $n$   $x$   $y$

*<proof>*

**theorem** *P2-secure*:

**assumes**  $x < q$   $n$   $y < q$   $n$

**shows** *sim-non-det-def.perfect-sec-P2*  $n$   $x$   $y$

*<proof>*

**end**

**end**

## 2.9 DHH Extension

We define a variant of the DDH assumption and show it is as hard as the original DDH assumption.

**theory** *DH-Ext imports*

*Game-Based-Crypto.Diffie-Hellman*

*Cyclic-Group-Ext*  
**begin**

**context** *ddh* **begin**

**definition** *DDH0* :: 'grp adversary  $\Rightarrow$  bool spmf  
**where** *DDH0*  $\mathcal{A} = do$  {  
 $s \leftarrow sample\text{-}uniform$  (order  $\mathcal{G}$ );  
 $r \leftarrow sample\text{-}uniform$  (order  $\mathcal{G}$ );  
 $let\ h = \mathbf{g} [\uparrow] s$ ;  
 $\mathcal{A}\ h (\mathbf{g} [\uparrow] r) (h [\uparrow] r)$ }

**definition** *DDH1* :: 'grp adversary  $\Rightarrow$  bool spmf  
**where** *DDH1*  $\mathcal{A} = do$  {  
 $s \leftarrow sample\text{-}uniform$  (order  $\mathcal{G}$ );  
 $r \leftarrow sample\text{-}uniform$  (order  $\mathcal{G}$ );  
 $let\ h = \mathbf{g} [\uparrow] s$ ;  
 $\mathcal{A}\ h (\mathbf{g} [\uparrow] r) ((h [\uparrow] r) \otimes \mathbf{g})$ }

**definition** *DDH-advantage* :: 'grp adversary  $\Rightarrow$  real  
**where** *DDH-advantage*  $\mathcal{A} = |spmf (DDH0\ \mathcal{A})\ True - spmf (DDH1\ \mathcal{A})\ True|$

**definition** *DDH-A'* :: 'grp adversary  $\Rightarrow$  'grp  $\Rightarrow$  'grp  $\Rightarrow$  'grp  $\Rightarrow$  bool spmf  
**where** *DDH-A'*  $D\text{-}ddh\ a\ b\ c = D\text{-}ddh\ a\ b\ (c \otimes \mathbf{g})$

**end**

**locale** *ddh-ext* = *ddh* + *cyclic-group*  $\mathcal{G}$   
**begin**

**lemma** *DDH0-eq-ddh-0*: *ddh.DDH0*  $\mathcal{G}\ \mathcal{A} = ddh.ddh\text{-}0\ \mathcal{G}\ \mathcal{A}$   
*<proof>*

**lemma** *DDH-bound1*:  $|spmf (ddh.DDH0\ \mathcal{G}\ \mathcal{A})\ True - spmf (ddh.DDH1\ \mathcal{G}\ \mathcal{A})\ True|$   
 $\leq |spmf (ddh.ddh\text{-}0\ \mathcal{G}\ \mathcal{A})\ True - spmf (ddh.ddh\text{-}1\ \mathcal{G}\ \mathcal{A})\ True|$   
 $+ |spmf (ddh.ddh\text{-}1\ \mathcal{G}\ \mathcal{A})\ True - spmf (ddh.DDH1\ \mathcal{G}\ \mathcal{A})\ True|$   
*<proof>*

**lemma** *DDH-bound2*:  
**shows**  $|spmf (ddh.DDH0\ \mathcal{G}\ \mathcal{A})\ True - spmf (ddh.DDH1\ \mathcal{G}\ \mathcal{A})\ True|$   
 $\leq ddh.advantage\ \mathcal{G}\ \mathcal{A} + |spmf (ddh.ddh\text{-}1\ \mathcal{G}\ \mathcal{A})\ True - spmf (ddh.DDH1\ \mathcal{G}\ \mathcal{A})\ True|$   
*<proof>*

**lemma** *rewrite*:  
**shows**  $(sample\text{-}uniform\ (order\ \mathcal{G}) \gg (\lambda x. sample\text{-}uniform\ (order\ \mathcal{G}))$   
 $\gg (\lambda y. sample\text{-}uniform\ (order\ \mathcal{G})) \gg (\lambda z. \mathcal{A}\ (\mathbf{g} [\uparrow] x)\ (\mathbf{g} [\uparrow] y)\ (\mathbf{g} [\uparrow] z)$



$\otimes \mathbf{g})\rangle\rangle\rangle\rangle)$   
 $= (\text{sample-uniform } (\text{order } \mathcal{G}) \gg (\lambda x. \text{sample-uniform } (\text{order } \mathcal{G})$   
 $\gg (\lambda y. \text{sample-uniform } (\text{order } \mathcal{G}) \gg (\lambda z. \mathcal{A} (\mathbf{g} [\ ] x) (\mathbf{g} [\ ] y) (\mathbf{g}$   
 $[\ ] z))))$   
 $(\text{is } ?lhs = ?rhs)$   
 $\langle \text{proof} \rangle$

**lemma** *DDH- $\mathcal{A}'$ -bound*:  $\text{ddh.}advantage \mathcal{G} (\text{ddh.DDH-}\mathcal{A}' \mathcal{G} \mathcal{A}) = |\text{spmf } (\text{ddh.ddh-1}$   
 $\mathcal{G} \mathcal{A}) \text{ True} - \text{spmf } (\text{ddh.DDH1 } \mathcal{G} \mathcal{A}) \text{ True}|$   
 $\langle \text{proof} \rangle$

**lemma** *DDH-advantage-bound*:  $\text{ddh.DDH-advantage } \mathcal{G} \mathcal{A} \leq \text{ddh.}advantage \mathcal{G} \mathcal{A} +$   
 $\text{ddh.}advantage \mathcal{G} (\text{ddh.DDH-}\mathcal{A}' \mathcal{G} \mathcal{A})$   
 $\langle \text{proof} \rangle$

**end**

**end**

### 3 Malicious Security

Here we define security in the malicious security setting. We follow the definitions given in [4] and [2]. The definition of malicious security follows the real/ideal world paradigm.

#### 3.1 Malicious Security Definitions

**theory** *Malicious-Defs* **imports**

*CryptHOL.CryptHOL*

**begin**

**type-synonym**  $(\text{'in1'}, \text{'aux'}, \text{'P1-S1-aux'}) \text{ P1-ideal-adv1} = \text{'in1'} \Rightarrow \text{'aux'} \Rightarrow (\text{'in1'}$   
 $\times \text{'P1-S1-aux'}) \text{ spmf}$

**type-synonym**  $(\text{'in1'}, \text{'aux'}, \text{'out1'}, \text{'P1-S1-aux'}, \text{'adv-out1'}) \text{ P1-ideal-adv2} = \text{'in1'}$   
 $\Rightarrow \text{'aux'} \Rightarrow \text{'out1'} \Rightarrow \text{'P1-S1-aux'} \Rightarrow \text{'adv-out1'} \text{ spmf}$

**type-synonym**  $(\text{'in1'}, \text{'aux'}, \text{'out1'}, \text{'P1-S1-aux'}, \text{'adv-out1'}) \text{ P1-ideal-adv} = (\text{'in1'}, \text{'aux'},$   
 $\text{'P1-S1-aux'}) \text{ P1-ideal-adv1} \times (\text{'in1'}, \text{'aux'}, \text{'out1'}, \text{'P1-S1-aux'}, \text{'adv-out1'}) \text{ P1-ideal-adv2}$

**type-synonym**  $(\text{'P1-real-adv'}, \text{'in1'}, \text{'aux'}, \text{'P1-S1-aux'}) \text{ P1-sim1} = \text{'P1-real-adv'}$   
 $\Rightarrow \text{'in1'} \Rightarrow \text{'aux'} \Rightarrow (\text{'in1'} \times \text{'P1-S1-aux'}) \text{ spmf}$

**type-synonym**  $(\text{'P1-real-adv'}, \text{'in1'}, \text{'aux'}, \text{'out1'}, \text{'P1-S1-aux'}, \text{'adv-out1'}) \text{ P1-sim2}$

$= \text{'P1-real-adv'} \Rightarrow \text{'in1'} \Rightarrow \text{'aux'} \Rightarrow \text{'out1'}$   
 $\Rightarrow \text{'P1-S1-aux'} \Rightarrow \text{'adv-out1'} \text{ spmf}$

**type-synonym** ('P1-real-adv', 'in1', 'aux', 'out1', 'P1-S1-aux', 'adv-out1') P1-sim  
= (('P1-real-adv', 'in1', 'aux', 'P1-S1-aux') P1-sim1  
× ('P1-real-adv', 'in1', 'aux', 'out1', 'P1-S1-aux', 'adv-out1')  
P1-sim2)

**type-synonym** ('in2', 'aux', 'P2-S2-aux') P2-ideal-adv1 = 'in2' ⇒ 'aux' ⇒ ('in2'  
× 'P2-S2-aux') spmf

**type-synonym** ('in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2') P2-ideal-adv2  
= 'in2' ⇒ 'aux' ⇒ 'out2' ⇒ 'P2-S2-aux' ⇒ 'adv-out2' spmf

**type-synonym** ('in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2') P2-ideal-adv  
= ('in2', 'aux', 'P2-S2-aux') P2-ideal-adv1  
× ('in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2') P2-ideal-adv2

**type-synonym** ('P2-real-adv', 'in2', 'aux', 'P2-S2-aux') P2-sim1 = 'P2-real-adv'  
⇒ 'in2' ⇒ 'aux' ⇒ ('in2' × 'P2-S2-aux') spmf

**type-synonym** ('P2-real-adv', 'in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2') P2-sim2  
= 'P2-real-adv' ⇒ 'in2' ⇒ 'aux' ⇒ 'out2'  
⇒ 'P2-S2-aux' ⇒ 'adv-out2' spmf

**type-synonym** ('P2-real-adv', 'in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2') P2-sim  
= (('P2-real-adv', 'in2', 'aux', 'P2-S2-aux') P2-sim1  
× ('P2-real-adv', 'in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2')  
P2-sim2)

**locale** malicious-base =

**fixes** *funct* :: 'in1 ⇒ 'in2 ⇒ ('out1 × 'out2) spmf — the functionality  
**and** *protocol* :: 'in1 ⇒ 'in2 ⇒ ('out1 × 'out2) spmf — outputs the output of  
each party in the protocol  
**and** *S1-1* :: ('P1-real-adv', 'in1', 'aux', 'P1-S1-aux') P1-sim1 — first part of the  
simulator for party 1  
**and** *S1-2* :: ('P1-real-adv', 'in1', 'aux', 'out1', 'P1-S1-aux', 'adv-out1') P1-sim2 —  
second part of the simulator for party 1  
**and** *P1-real-view* :: 'in1 ⇒ 'in2 ⇒ 'aux ⇒ 'P1-real-adv ⇒ ('adv-out1 × 'out2)  
spmf — real view for party 1, the adversary totally controls party 1  
**and** *S2-1* :: ('P2-real-adv', 'in2', 'aux', 'P2-S2-aux') P2-sim1 — first part of the  
simulator for party 2  
**and** *S2-2* :: ('P2-real-adv', 'in2', 'aux', 'out2', 'P2-S2-aux', 'adv-out2') P2-sim2 —  
second part of the simulator for party 1  
**and** *P2-real-view* :: 'in1 ⇒ 'in2 ⇒ 'aux ⇒ 'P2-real-adv ⇒ ('out1 × 'adv-out2)  
spmf — real view for party 2, the adversary totally controls party 2  
**begin**

**definition** *correct* *m1* *m2* ↔ (protocol *m1* *m2* = *funct* *m1* *m2*)

**abbreviation**  $\text{trusted-party } x y \equiv \text{funct } x y$

The ideal game defines the ideal world. First we consider the case where party 1 is corrupt, and thus controlled by the adversary. The adversary is split into two parts, the first part takes the original input and auxiliary information and outputs its input to the protocol. The trusted party then computes the functionality on the input given by the adversary and the correct input for party 2. Party 2 outputs the its correct output as given by the trusted party, the adversary provides the output for party 1.

**definition**  $\text{ideal-game-1} :: 'in1 \Rightarrow 'in2 \Rightarrow 'aux \Rightarrow ('in1, 'aux, 'out1, 'P1-S1-aux, 'adv-out1) P1\text{-ideal-adv} \Rightarrow ('adv-out1 \times 'out2) \text{ spmf}$

**where**  $\text{ideal-game-1 } x y z A = \text{do } \{$   
 $\text{let } (A1, A2) = A;$   
 $(x', \text{aux-out}) \leftarrow A1 x z;$   
 $(out1, out2) \leftarrow \text{trusted-party } x' y;$   
 $out1' :: 'adv-out1 \leftarrow A2 x' z out1 \text{ aux-out};$   
 $\text{return-spmf } (out1', out2)\}$

**definition**  $\text{ideal-view-1} :: 'in1 \Rightarrow 'in2 \Rightarrow 'aux \Rightarrow ('P1\text{-real-adv}, 'in1, 'aux, 'out1, 'P1-S1-aux, 'adv-out1) P1\text{-sim} \Rightarrow 'P1\text{-real-adv} \Rightarrow ('adv-out1 \times 'out2) \text{ spmf}$

**where**  $\text{ideal-view-1 } x y z S \mathcal{A} = (\text{let } (S1, S2) = S \text{ in } (\text{ideal-game-1 } x y z (S1 \ \mathcal{A}, S2 \ \mathcal{A})))$

We have information theoretic security when the real and ideal views are equal.

**definition**  $\text{perfect-sec-P1 } x y z S \mathcal{A} \longleftrightarrow (\text{ideal-view-1 } x y z S \mathcal{A} = P1\text{-real-view } x y z \mathcal{A})$

The advantage of party 1 denotes the probability of a distinguisher distinguishing the real and ideal views.

**definition**  $\text{adv-P1 } x y z S \mathcal{A} (D :: ('adv-out1 \times 'out2) \Rightarrow \text{bool } \text{ spmf}) =$   
 $|\text{spmf } (P1\text{-real-view } x y z \mathcal{A} \gg (\lambda \text{ view. } D \text{ view})) \text{ True}$   
 $- \text{spmf } (\text{ideal-view-1 } x y z S \mathcal{A} \gg (\lambda \text{ view. } D \text{ view})) \text{ True }|$

**definition**  $\text{ideal-game-2} :: 'in1 \Rightarrow 'in2 \Rightarrow 'aux \Rightarrow ('in2, 'aux, 'out2, 'P2-S2-aux, 'adv-out2) P2\text{-ideal-adv} \Rightarrow ('out1 \times 'adv-out2) \text{ spmf}$

**where**  $\text{ideal-game-2 } x y z A = \text{do } \{$   
 $\text{let } (A1, A2) = A;$   
 $(y', \text{aux-out}) \leftarrow A1 y z;$   
 $(out1, out2) \leftarrow \text{trusted-party } x y';$   
 $out2' :: 'adv-out2 \leftarrow A2 y' z out2 \text{ aux-out};$   
 $\text{return-spmf } (out1, out2')\}$

**definition**  $\text{ideal-view-2} :: 'in1 \Rightarrow 'in2 \Rightarrow 'aux \Rightarrow ('P2\text{-real-adv}, 'in2, 'aux, 'out2, 'P2-S2-aux, 'adv-out2) P2\text{-sim} \Rightarrow 'P2\text{-real-adv} \Rightarrow ('out1 \times 'adv-out2) \text{ spmf}$

**where**  $\text{ideal-view-2 } x y z S \mathcal{A} = (\text{let } (S1, S2) = S \text{ in } (\text{ideal-game-2 } x y z (S1 \ \mathcal{A}, S2 \ \mathcal{A})))$

**definition** *perfect-sec-P2*  $x\ y\ z\ S\ \mathcal{A} \longleftrightarrow (\text{ideal-view-2 } x\ y\ z\ S\ \mathcal{A} = \text{P2-real-view } x\ y\ z\ \mathcal{A})$

**definition** *adv-P2*  $x\ y\ z\ S\ \mathcal{A} (D :: ('out1 \times 'adv-out2) \Rightarrow \text{bool } \text{spmf}) =$   
 $| \text{spmf } (\text{P2-real-view } x\ y\ z\ \mathcal{A} \gg= (\lambda \text{ view. } D \text{ view})) \text{ True}$   
 $- \text{spmf } (\text{ideal-view-2 } x\ y\ z\ S\ \mathcal{A} \gg= (\lambda \text{ view. } D \text{ view})) \text{ True } |$

**end**

**end**

### 3.2 Malicious OT

Here we prove secure the 1-out-of-2 OT protocol given in [4] (p190). For party 1 reduce security to the DDH assumption and for party 2 we show information theoretic security.

**theory** *Malicious-OT imports*

*HOL-Number-Theory.Cong*

*Cyclic-Group-Ext*

*DH-Ext*

*Malicious-Defs*

*Number-Theory-Aux*

*OT-Functionalities*

*Uniform-Sampling*

**begin**

**type-synonym**  $('aux, 'grp', 'state) \text{adv-1-P1} = ('grp' \times 'grp') \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow 'aux \Rightarrow (('grp' \times 'grp' \times 'grp') \times 'state) \text{spmf}$

**type-synonym**  $('grp', 'state) \text{adv-2-P1} = 'grp' \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow 'grp' \Rightarrow ('grp' \times 'grp') \Rightarrow 'state \Rightarrow ((('grp' \times 'grp') \times ('grp' \times 'grp')) \times 'state) \text{spmf}$

**type-synonym**  $('adv-out1, 'state) \text{adv-3-P1} = 'state \Rightarrow 'adv-out1 \text{spmf}$

**type-synonym**  $('aux, 'grp', 'adv-out1, 'state) \text{adv-mal-P1} = (('aux, 'grp', 'state) \text{adv-1-P1} \times ('grp', 'state) \text{adv-2-P1} \times ('adv-out1, 'state) \text{adv-3-P1})$

**type-synonym**  $('aux, 'grp', 'state) \text{adv-1-P2} = \text{bool} \Rightarrow 'aux \Rightarrow (('grp' \times 'grp' \times 'grp' \times 'grp' \times 'grp') \times 'state) \text{spmf}$

**type-synonym**  $('grp', 'state) \text{adv-2-P2} = ('grp' \times 'grp' \times 'grp' \times 'grp' \times 'grp') \Rightarrow 'state \Rightarrow ((('grp' \times 'grp' \times 'grp') \times \text{nat}) \times 'state) \text{spmf}$

**type-synonym**  $('grp', 'adv-out2, 'state) \text{adv-3-P2} = ('grp' \times 'grp') \Rightarrow ('grp' \times 'grp') \Rightarrow 'state \Rightarrow 'adv-out2 \text{spmf}$

**type-synonym** ('aux, 'grp', 'adv-out2, 'state) *adv-mal-P2* = (('aux, 'grp', 'state) *adv-1-P2* × ('grp', 'state) *adv-2-P2* × ('grp', 'adv-out2, 'state) *adv-3-P2*)

**locale** *ot-base* =

**fixes**  $\mathcal{G} :: 'grp$  *cyclic-group* (**structure**)  
**assumes** *finite-group: finite* (*carrier*  $\mathcal{G}$ )  
**and** *order-gt-0: order*  $\mathcal{G} > 0$   
**and** *prime-order: prime* (*order*  $\mathcal{G}$ )

**begin**

**lemma** *prime-field: a < (order*  $\mathcal{G}$ )  $\implies a \neq 0 \implies$  *coprime a (order*  $\mathcal{G}$ )

*<proof>*

The protocol uses a call to an idealised functionality of a zero knowledge protocol for the DDH relation, this is described by the functionality given below.

**fun** *funct-DH-ZK* :: ('grp × 'grp × 'grp)  $\Rightarrow$  (('grp × 'grp × 'grp) × nat)  $\Rightarrow$  (bool × unit) *spmf*  
**where** *funct-DH-ZK* (h,a,b) ((h',a',b'),r) = *return-spmf* (a = **g** [∧] r ∧ b = h [∧] r ∧ (h,a,b) = (h',a',b'), ())

The probabilistic program that defines the output for both parties in the protocol.

**definition** *protocol-ot* :: ('grp × 'grp)  $\Rightarrow$  bool  $\Rightarrow$  (unit × 'grp) *spmf*  
**where** *protocol-ot* M  $\sigma$  = *do* {  
*let* (x0,x1) = M;  
*r*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*α0*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*α1*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*let* h0 = **g** [∧] *α0*;  
*let* h1 = **g** [∧] *α1*;  
*let* a = **g** [∧] *r*;  
*let* b0 = h0 [∧] *r* ⊗ **g** [∧] (*if*  $\sigma$  *then* (1::nat) *else* 0);  
*let* b1 = h1 [∧] *r* ⊗ **g** [∧] (*if*  $\sigma$  *then* (1::nat) *else* 0);  
*let* h = h0 ⊗ *inv* h1;  
*let* b = b0 ⊗ *inv* b1;  
- :: unit  $\leftarrow$  *assert-spmf* (a = **g** [∧] *r* ∧ b = h [∧] *r*);  
*u0*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*u1*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*v0*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*v1*  $\leftarrow$  *sample-uniform* (*order*  $\mathcal{G}$ );  
*let* z0 = b0 [∧] *u0* ⊗ h0 [∧] *v0* ⊗ x0;  
*let* w0 = a [∧] *u0* ⊗ **g** [∧] *v0*;  
*let* e0 = (w0, z0);  
*let* z1 = (b1 ⊗ *inv* **g**) [∧] *u1* ⊗ h1 [∧] *v1* ⊗ x1;  
*let* w1 = a [∧] *u1* ⊗ **g** [∧] *v1*;  
*let* e1 = (w1, z1);  
*return-spmf* ((), (*if*  $\sigma$  *then* (z1 ⊗ *inv* (w1 [∧] *α1*)) *else* (z0 ⊗ *inv* (w0 [∧] *α0*))))}

Party 1 sends three messages (including the output) in the protocol so we split the adversary into three parts, one part to output each message. The real view of the protocol for party 1 outputs the correct output for party 2 and the adversary outputs the output for party 1.

**definition**  $P1\text{-real-model} :: ('grp \times 'grp) \Rightarrow \text{bool} \Rightarrow 'aux \Rightarrow ('aux, 'grp, 'adv\text{-out}1, 'state) \text{adv-mal-P1} \Rightarrow ('adv\text{-out}1 \times 'grp) \text{spmf}$

**where**  $P1\text{-real-model } M \sigma z \mathcal{A} = \text{do} \{$   
 $\text{let } (\mathcal{A}1, \mathcal{A}2, \mathcal{A}3) = \mathcal{A};$   
 $r \leftarrow \text{sample-uniform } (\text{order } \mathcal{G});$   
 $\alpha0 \leftarrow \text{sample-uniform } (\text{order } \mathcal{G});$   
 $\alpha1 \leftarrow \text{sample-uniform } (\text{order } \mathcal{G});$   
 $\text{let } h0 = \mathbf{g} [\uparrow] \alpha0;$   
 $\text{let } h1 = \mathbf{g} [\uparrow] \alpha1;$   
 $\text{let } a = \mathbf{g} [\uparrow] r;$   
 $\text{let } b0 = h0 [\uparrow] r \otimes (\text{if } \sigma \text{ then } \mathbf{g} \text{ else } \mathbf{1});$   
 $\text{let } b1 = h1 [\uparrow] r \otimes (\text{if } \sigma \text{ then } \mathbf{g} \text{ else } \mathbf{1});$   
 $((in1 :: 'grp, in2 :: 'grp, in3 :: 'grp), s) \leftarrow \mathcal{A}1 M h0 h1 a b0 b1 z;$   
 $\text{let } (h,a,b) = (h0 \otimes \text{inv } h1, a, b0 \otimes \text{inv } b1);$   
 $(b :: \text{bool}, - :: \text{unit}) \leftarrow \text{funct-DH-ZK } (in1, in2, in3) ((h,a,b), r);$   
 $- :: \text{unit} \leftarrow \text{assert-spmf } (b);$   
 $((w0,z0),(w1,z1), s') \leftarrow \mathcal{A}2 h0 h1 a b0 b1 M s;$   
 $\text{adv-out} :: 'adv\text{-out}1 \leftarrow \mathcal{A}3 s';$   
 $\text{return-spmf } (\text{adv-out}, (\text{if } \sigma \text{ then } (z1 \otimes (\text{inv } w1 [\uparrow] \alpha1)) \text{ else } (z0 \otimes (\text{inv } w0 [\uparrow] \alpha0))))\}$

The first and second part of the simulator for party 1 are defined below.

**definition**  $P1\text{-S1} :: ('aux, 'grp, 'adv\text{-out}1, 'state) \text{adv-mal-P1} \Rightarrow ('grp \times 'grp) \Rightarrow 'aux \Rightarrow (('grp \times 'grp) \times 'state) \text{spmf}$

**where**  $P1\text{-S1 } \mathcal{A} M z = \text{do} \{$   
 $\text{let } (\mathcal{A}1, \mathcal{A}2, \mathcal{A}3) = \mathcal{A};$   
 $r \leftarrow \text{sample-uniform } (\text{order } \mathcal{G});$   
 $\alpha0 \leftarrow \text{sample-uniform } (\text{order } \mathcal{G});$   
 $\alpha1 \leftarrow \text{sample-uniform } (\text{order } \mathcal{G});$   
 $\text{let } h0 = \mathbf{g} [\uparrow] \alpha0;$   
 $\text{let } h1 = \mathbf{g} [\uparrow] \alpha1;$   
 $\text{let } a = \mathbf{g} [\uparrow] r;$   
 $\text{let } b0 = h0 [\uparrow] r;$   
 $\text{let } b1 = h1 [\uparrow] r \otimes \mathbf{g};$   
 $((in1 :: 'grp, in2 :: 'grp, in3 :: 'grp), s) \leftarrow \mathcal{A}1 M h0 h1 a b0 b1 z;$   
 $\text{let } (h,a,b) = (h0 \otimes \text{inv } h1, a, b0 \otimes \text{inv } b1);$   
 $- :: \text{unit} \leftarrow \text{assert-spmf } ((in1, in2, in3) = (h,a,b));$   
 $((w0,z0),(w1,z1), s') \leftarrow \mathcal{A}2 h0 h1 a b0 b1 M s;$   
 $\text{let } x0 = (z0 \otimes (\text{inv } w0 [\uparrow] \alpha0));$   
 $\text{let } x1 = (z1 \otimes (\text{inv } w1 [\uparrow] \alpha1));$   
 $\text{return-spmf } ((x0,x1), s')\}$

**definition**  $P1\text{-S2} :: ('aux, 'grp, 'adv\text{-out}1, 'state) \text{adv-mal-P1} \Rightarrow ('grp \times 'grp) \Rightarrow 'aux \Rightarrow \text{unit} \Rightarrow 'state \Rightarrow 'adv\text{-out}1 \text{spmf}$

**where**  $P1\text{-S2 } \mathcal{A} M z \text{out}1 s' = \text{do} \{$

let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;  
 $\mathcal{A}3$   $s'$ }

We explicitly provide the unfolded definition of the ideal model for convenience in the proof.

**definition**  $P1$ -ideal-model :: ('grp × 'grp) ⇒ bool ⇒ 'aux ⇒ ('aux, 'grp, 'adv-out1, 'state)  
adv-mal-P1 ⇒ ('adv-out1 × 'grp) spmf  
**where**  $P1$ -ideal-model  $M$   $\sigma$   $z$   $\mathcal{A}$  = do {  
let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;  
 $r$  ← sample-uniform (order  $\mathcal{G}$ );  
 $\alpha0$  ← sample-uniform (order  $\mathcal{G}$ );  
 $\alpha1$  ← sample-uniform (order  $\mathcal{G}$ );  
let  $h0$  =  $\mathbf{g}$  [ $\checkmark$ ]  $\alpha0$ ;  
let  $h1$  =  $\mathbf{g}$  [ $\checkmark$ ]  $\alpha1$ ;  
let  $a$  =  $\mathbf{g}$  [ $\checkmark$ ]  $r$ ;  
let  $b0$  =  $h0$  [ $\checkmark$ ]  $r$ ;  
let  $b1$  =  $h1$  [ $\checkmark$ ]  $r$   $\otimes$   $\mathbf{g}$ ;  
(( $in1$  :: 'grp,  $in2$  :: 'grp,  $in3$  :: 'grp),  $s$ ) ←  $\mathcal{A}1$   $M$   $h0$   $h1$   $a$   $b0$   $b1$   $z$ ;  
let ( $h,a,b$ ) = ( $h0$   $\otimes$   $inv$   $h1$ ,  $a$ ,  $b0$   $\otimes$   $inv$   $b1$ );  
- :: unit ← assert-spmf (( $in1$ ,  $in2$ ,  $in3$ ) = ( $h,a,b$ ));  
((( $w0,z0$ ),( $w1,z1$ )), $s'$ ) ←  $\mathcal{A}2$   $h0$   $h1$   $a$   $b0$   $b1$   $M$   $s$ ;  
let  $x0'$  =  $z0$   $\otimes$   $inv$   $w0$  [ $\checkmark$ ]  $\alpha0$ ;  
let  $x1'$  =  $z1$   $\otimes$   $inv$   $w1$  [ $\checkmark$ ]  $\alpha1$ ;  
(-,  $f$ -out2) ← funct-OT-12 ( $x0'$ ,  $x1'$ )  $\sigma$ ;  
adv-out :: 'adv-out1 ←  $\mathcal{A}3$   $s'$ ;  
return-spmf (adv-out,  $f$ -out2)}

The advantage associated with the unfolded definition of the ideal view.

**definition**

$P1$ -adv-real-ideal-model ( $D$  :: ('adv-out1 × 'grp) ⇒ bool spmf)  $M$   $\sigma$   $\mathcal{A}$   $z$   
= |spmf (( $P1$ -real-model  $M$   $\sigma$   $z$   $\mathcal{A}$ )  $\gg$  (λ view.  $D$  view)) True  
- spmf (( $P1$ -ideal-model  $M$   $\sigma$   $z$   $\mathcal{A}$ )  $\gg$  (λ view.  $D$  view))

True|

We now define the real view and simulators for party 2 in an analogous way.

**definition**  $P2$ -real-model :: ('grp × 'grp) ⇒ bool ⇒ 'aux ⇒ ('aux, 'grp, 'adv-out2, 'state)  
adv-mal-P2 ⇒ (unit × 'adv-out2) spmf  
**where**  $P2$ -real-model  $M$   $\sigma$   $z$   $\mathcal{A}$  = do {  
let ( $x0,x1$ ) =  $M$ ;  
let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;  
(( $h0,h1,a,b0,b1$ ), $s$ ) ←  $\mathcal{A}1$   $\sigma$   $z$ ;  
- :: unit ← assert-spmf ( $h0$  ∈ carrier  $\mathcal{G}$  ∧  $h1$  ∈ carrier  $\mathcal{G}$  ∧  $a$  ∈ carrier  $\mathcal{G}$  ∧  
 $b0$  ∈ carrier  $\mathcal{G}$  ∧  $b1$  ∈ carrier  $\mathcal{G}$ );  
((( $in1$ ,  $in2$ ,  $in3$  :: 'grp),  $r$ ), $s'$ ) ←  $\mathcal{A}2$  ( $h0,h1,a,b0,b1$ )  $s$ ;  
let ( $h,a,b$ ) = ( $h0$   $\otimes$   $inv$   $h1$ ,  $a$ ,  $b0$   $\otimes$   $inv$   $b1$ );  
(out-zk-funct, -) ← funct-DH-ZK ( $h,a,b$ ) (( $in1$ ,  $in2$ ,  $in3$ ),  $r$ );  
- :: unit ← assert-spmf out-zk-funct;  
 $u0$  ← sample-uniform (order  $\mathcal{G}$ );  
 $u1$  ← sample-uniform (order  $\mathcal{G}$ );

```

v0 ← sample-uniform (order  $\mathcal{G}$ );
v1 ← sample-uniform (order  $\mathcal{G}$ );
let z0 = b0 [∧] u0 ⊗ h0 [∧] v0 ⊗ x0;
let w0 = a [∧] u0 ⊗ g [∧] v0;
let e0 = (w0, z0);
let z1 = (b1 ⊗ inv g) [∧] u1 ⊗ h1 [∧] v1 ⊗ x1;
let w1 = a [∧] u1 ⊗ g [∧] v1;
let e1 = (w1, z1);
out ←  $\mathcal{A}3$  e0 e1 s';
return-spmf ((), out)}

```

**definition**  $P2-S1 :: ('aux, 'grp, 'adv-out2, 'state) adv-mal-P2 \Rightarrow bool \Rightarrow 'aux \Rightarrow (bool \times ('grp \times 'grp \times 'grp \times 'grp \times 'grp) \times 'state) \text{ spmf}$   
**where**  $P2-S1 \mathcal{A} \sigma z = do \{$   
 let  $(\mathcal{A}1, \mathcal{A}2, \mathcal{A}3) = \mathcal{A};$   
 $((h0, h1, a, b0, b1), s) \leftarrow \mathcal{A}1 \sigma z;$   
 $- :: unit \leftarrow assert-spmf (h0 \in carrier \mathcal{G} \wedge h1 \in carrier \mathcal{G} \wedge a \in carrier \mathcal{G} \wedge b0 \in carrier \mathcal{G} \wedge b1 \in carrier \mathcal{G});$   
 $((in1, in2, in3 :: 'grp), r), s' \leftarrow \mathcal{A}2 (h0, h1, a, b0, b1) s;$   
 let  $(h, a, b) = (h0 \otimes inv h1, a, b0 \otimes inv b1);$   
 $(out-zk-funct, -) \leftarrow funct-DH-ZK (h, a, b) ((in1, in2, in3), r);$   
 $- :: unit \leftarrow assert-spmf out-zk-funct;$   
 let  $l = b0 \otimes (inv (h0 [∧] r));$   
 $return-spmf ((if l = \mathbf{1} \text{ then False else True), (h0, h1, a, b0, b1), s')\}$

**definition**  $P2-S2 :: ('aux, 'grp, 'adv-out2, 'state) adv-mal-P2 \Rightarrow bool \Rightarrow 'aux \Rightarrow 'grp \Rightarrow (('grp \times 'grp \times 'grp \times 'grp \times 'grp) \times 'state) \Rightarrow 'adv-out2 \text{ spmf}$   
**where**  $P2-S2 \mathcal{A} \sigma' z x\sigma aux-out = do \{$   
 let  $(\mathcal{A}1, \mathcal{A}2, \mathcal{A}3) = \mathcal{A};$   
 let  $((h0, h1, a, b0, b1), s) = aux-out;$   
 $u0 \leftarrow sample-uniform (order \mathcal{G});$   
 $v0 \leftarrow sample-uniform (order \mathcal{G});$   
 $u1 \leftarrow sample-uniform (order \mathcal{G});$   
 $v1 \leftarrow sample-uniform (order \mathcal{G});$   
 let  $w0 = a [∧] u0 \otimes g [∧] v0;$   
 let  $w1 = a [∧] u1 \otimes g [∧] v1;$   
 let  $z0 = b0 [∧] u0 \otimes h0 [∧] v0 \otimes (if \sigma' \text{ then } \mathbf{1} \text{ else } x\sigma);$   
 let  $z1 = (b1 \otimes inv g) [∧] u1 \otimes h1 [∧] v1 \otimes (if \sigma' \text{ then } x\sigma \text{ else } \mathbf{1});$   
 let  $e0 = (w0, z0);$   
 let  $e1 = (w1, z1);$   
 $\mathcal{A}3 e0 e1 s\}$

**sublocale**  $mal-def : malicious-base funct-OT-12 protocol-ot P1-S1 P1-S2 P1-real-model P2-S1 P2-S2 P2-real-model \langle proof \rangle$

We prove the unfolded definition of the ideal views are equal to the definition we provide in the abstract locale that defines security.

**lemma**  $P1-ideal-ideal-eq:$

**shows**  $mal-def.ideal-view-1 x y z (P1-S1, P1-S2) \mathcal{A} = P1-ideal-model x y z \mathcal{A}$



**including** *monad-normalisation*  
 ⟨*proof*⟩

**lemma** *P1-advantages-eq*:

**shows** *mal-def.adv-P1*  $x\ y\ z\ (P1-S1, P1-S2)\ \mathcal{A}\ D = P1\text{-adv-real-ideal-model}\ D$   
 $x\ y\ \mathcal{A}\ z$   
 ⟨*proof*⟩

**fun** *P1-DDH-mal-adv-σ-false* :: ('grp × 'grp) ⇒ 'aux ⇒ ('aux, 'grp, 'adv-out1, 'state)  
*adv-mal-P1* ⇒ (('adv-out1 × 'grp) ⇒ bool spmf) ⇒ 'grp ddh.adversary  
**where** *P1-DDH-mal-adv-σ-false*  $M\ z\ \mathcal{A}\ D\ h\ a\ t = do\ \{\$   
 let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;  
 $\alpha0 \leftarrow sample\text{-uniform}\ (order\ \mathcal{G})$ ;  
 let  $h0 = \mathbf{g}\ [\checkmark]\ \alpha0$ ;  
 let  $h1 = h$ ;  
 let  $b0 = a\ [\checkmark]\ \alpha0$ ;  
 let  $b1 = t$ ;  
 (( $in1 :: 'grp, in2 :: 'grp, in3 :: 'grp$ ),  $s$ ) ←  $\mathcal{A}1\ M\ h0\ h1\ a\ b0\ b1\ z$ ;  
 - ::  $unit \leftarrow assert\text{-spmf}\ (in1 = h0 \otimes inv\ h1 \wedge in2 = a \wedge in3 = b0 \otimes inv\ b1)$ ;  
 (( $(w0, z0), (w1, z1)$ ),  $s'$ ) ←  $\mathcal{A}2\ h0\ h1\ a\ b0\ b1\ M\ s$ ;  
 let  $x0 = (z0 \otimes (inv\ w0\ [\checkmark]\ \alpha0))$ ;  
 $adv\text{-out} :: 'adv\text{-out}1 \leftarrow \mathcal{A}3\ s'$ ;  
 $D\ (adv\text{-out}, x0)\}$

**fun** *P1-DDH-mal-adv-σ-true* :: ('grp × 'grp) ⇒ 'aux ⇒ ('aux, 'grp, 'adv-out1, 'state)  
*adv-mal-P1* ⇒ (('adv-out1 × 'grp) ⇒ bool spmf) ⇒ 'grp ddh.adversary  
**where** *P1-DDH-mal-adv-σ-true*  $M\ z\ \mathcal{A}\ D\ h\ a\ t = do\ \{\$   
 let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;  
 $\alpha1 :: nat \leftarrow sample\text{-uniform}\ (order\ \mathcal{G})$ ;  
 let  $h1 = \mathbf{g}\ [\checkmark]\ \alpha1$ ;  
 let  $h0 = h$ ;  
 let  $b0 = t$ ;  
 let  $b1 = a\ [\checkmark]\ \alpha1 \otimes \mathbf{g}$ ;  
 (( $in1 :: 'grp, in2 :: 'grp, in3 :: 'grp$ ),  $s$ ) ←  $\mathcal{A}1\ M\ h0\ h1\ a\ b0\ b1\ z$ ;  
 - ::  $unit \leftarrow assert\text{-spmf}\ (in1 = h0 \otimes inv\ h1 \wedge in2 = a \wedge in3 = b0 \otimes inv\ b1)$ ;  
 (( $(w0, z0), (w1, z1)$ ),  $s'$ ) ←  $\mathcal{A}2\ h0\ h1\ a\ b0\ b1\ M\ s$ ;  
 let  $x1 = (z1 \otimes (inv\ w1\ [\checkmark]\ \alpha1))$ ;  
 $adv\text{-out} :: 'adv\text{-out}1 \leftarrow \mathcal{A}3\ s'$ ;  
 $D\ (adv\text{-out}, x1)\}$

**definition** *P2-ideal-model* :: ('grp × 'grp) ⇒ bool ⇒ 'aux ⇒ ('aux, 'grp, 'adv-out2,  
 'state) *adv-mal-P2* ⇒ (unit × 'adv-out2) spmf

**where** *P2-ideal-model*  $M\ \sigma\ z\ \mathcal{A} = do\ \{\$   
 let ( $x0, x1$ ) =  $M$ ;  
 let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;  
 (( $h0, h1, a, b0, b1$ ),  $s$ ) ←  $\mathcal{A}1\ \sigma\ z$ ;  
 - ::  $unit \leftarrow assert\text{-spmf}\ (h0 \in carrier\ \mathcal{G} \wedge h1 \in carrier\ \mathcal{G} \wedge a \in carrier\ \mathcal{G} \wedge$   
 $b0 \in carrier\ \mathcal{G} \wedge b1 \in carrier\ \mathcal{G})$ ;  
 (( $in1, in2, in3$ ),  $r$ ),  $s'$ ) ←  $\mathcal{A}2\ (h0, h1, a, b0, b1)\ s$ ;

```

let (h,a,b) = (h0 ⊗ inv h1, a, b0 ⊗ inv b1);
(out-zk-funct, -) ← funct-DH-ZK (h,a,b) ((in1, in2, in3), r);
- :: unit ← assert-spmf out-zk-funct;
let l = b0 ⊗ (inv (h0 [∧] r));
let σ' = (if l = 1 then False else True);
(- :: unit, xσ) ← funct-OT-12 (x0, x1) σ';
u0 ← sample-uniform (order G);
v0 ← sample-uniform (order G);
u1 ← sample-uniform (order G);
v1 ← sample-uniform (order G);
let w0 = a [∧] u0 ⊗ g [∧] v0;
let w1 = a [∧] u1 ⊗ g [∧] v1;
let z0 = b0 [∧] u0 ⊗ h0 [∧] v0 ⊗ (if σ' then 1 else xσ);
let z1 = (b1 ⊗ inv g) [∧] u1 ⊗ h1 [∧] v1 ⊗ (if σ' then xσ else 1);
let e0 = (w0,z0);
let e1 = (w1,z1);
out ← A3 e0 e1 s';
return-spmf ((), out)}

```

**definition**  $P2\text{-ideal-model-end} :: ('grp \times 'grp) \Rightarrow 'grp \Rightarrow (('grp \times 'grp \times 'grp \times 'grp \times 'grp) \times 'state)$

$\Rightarrow ('grp, 'adv\text{-out}2, 'state) \text{adv-3-P2} \Rightarrow (unit \times$

$'adv\text{-out}2) \text{spmf}$

**where**  $P2\text{-ideal-model-end } M \text{ l bs } \mathcal{A}3 = do \{$

```

let (x0,x1) = M;
let ((h0,h1,a,b0,b1),s) = bs;
let σ' = (if l = 1 then False else True);
(- :: unit, xσ) ← funct-OT-12 (x0, x1) σ';
u0 ← sample-uniform (order G);
v0 ← sample-uniform (order G);
u1 ← sample-uniform (order G);
v1 ← sample-uniform (order G);
let w0 = a [∧] u0 ⊗ g [∧] v0;
let w1 = a [∧] u1 ⊗ g [∧] v1;
let z0 = b0 [∧] u0 ⊗ h0 [∧] v0 ⊗ (if σ' then 1 else xσ);
let z1 = (b1 ⊗ inv g) [∧] u1 ⊗ h1 [∧] v1 ⊗ (if σ' then xσ else 1);
let e0 = (w0,z0);
let e1 = (w1,z1);
out ← A3 e0 e1 s;
return-spmf ((), out)}

```

**definition**  $P2\text{-ideal-model}' :: ('grp \times 'grp) \Rightarrow bool \Rightarrow 'aux \Rightarrow ('aux, 'grp, 'adv\text{-out}2, 'state) \text{adv-mal-P2} \Rightarrow (unit \times 'adv\text{-out}2) \text{spmf}$

**where**  $P2\text{-ideal-model}' M \sigma z \mathcal{A} = do \{$

```

let (x0,x1) = M;
let (A1, A2, A3) = A;
((h0,h1,a,b0,b1),s) ← A1 σ z;
- :: unit ← assert-spmf (h0 ∈ carrier G ∧ h1 ∈ carrier G ∧ a ∈ carrier G ∧
b0 ∈ carrier G ∧ b1 ∈ carrier G);

```

```

(((in1, in2, in3 :: 'grp), r), s') ←  $\mathcal{A}2$  (h0, h1, a, b0, b1) s;
let (h, a, b) = (h0 ⊗ inv h1, a, b0 ⊗ inv b1);
(out-zk-funct, -) ← funct-DH-ZK (h, a, b) ((in1, in2, in3), r);
- :: unit ← assert-spmf out-zk-funct;
let l = b0 ⊗ (inv (h0 [∧] r));
P2-ideal-model-end (x0, x1) l ((h0, h1, a, b0, b1), s')  $\mathcal{A}3$ 

```

**lemma** *P2-ideal-model-rewrite*:  $P2\text{-ideal-model } M \sigma z \mathcal{A} = P2\text{-ideal-model}' M \sigma z \mathcal{A}$

*<proof>*

**definition** *P2-real-model-end* :: ('grp × 'grp) ⇒ (('grp × 'grp × 'grp × 'grp × 'grp × 'state) ⇒ ('grp, 'adv-out2, 'state) adv-3-P2 ⇒ (unit × 'adv-out2) spmf

```

where P2-real-model-end M bs  $\mathcal{A}3 = do$  {
  let (x0, x1) = M;
  let ((h0, h1, a, b0, b1), s) = bs;
  u0 ← sample-uniform (order  $\mathcal{G}$ );
  u1 ← sample-uniform (order  $\mathcal{G}$ );
  v0 ← sample-uniform (order  $\mathcal{G}$ );
  v1 ← sample-uniform (order  $\mathcal{G}$ );
  let z0 = b0 [∧] u0 ⊗ h0 [∧] v0 ⊗ x0;
  let w0 = a [∧] u0 ⊗ g [∧] v0;
  let e0 = (w0, z0);
  let z1 = (b1 ⊗ inv g) [∧] u1 ⊗ h1 [∧] v1 ⊗ x1;
  let w1 = a [∧] u1 ⊗ g [∧] v1;
  let e1 = (w1, z1);
  out ←  $\mathcal{A}3$  e0 e1 s;
  return-spmf ((), out)}

```

**definition** *P2-real-model'* :: ('grp × 'grp) ⇒ bool ⇒ 'aux ⇒ ('aux, 'grp, 'adv-out2, 'state) adv-mal-P2 ⇒ (unit × 'adv-out2) spmf

```

where P2-real-model' M  $\sigma z \mathcal{A} = do$  {
  let (x0, x1) = M;
  let ( $\mathcal{A}1, \mathcal{A}2, \mathcal{A}3$ ) =  $\mathcal{A}$ ;
  ((h0, h1, a, b0, b1), s) ←  $\mathcal{A}1$   $\sigma z$ ;
  - :: unit ← assert-spmf (h0 ∈ carrier  $\mathcal{G} \wedge h1 \in \text{carrier } \mathcal{G} \wedge a \in \text{carrier } \mathcal{G} \wedge b0 \in \text{carrier } \mathcal{G} \wedge b1 \in \text{carrier } \mathcal{G}$ );
  (((in1, in2, in3 :: 'grp), r), s') ←  $\mathcal{A}2$  (h0, h1, a, b0, b1) s;
  let (h, a, b) = (h0 ⊗ inv h1, a, b0 ⊗ inv b1);
  (out-zk-funct, -) ← funct-DH-ZK (h, a, b) ((in1, in2, in3), r);
  - :: unit ← assert-spmf out-zk-funct;
  P2-real-model-end M ((h0, h1, a, b0, b1), s')  $\mathcal{A}3$ 

```

**lemma** *P2-real-model-rewrite*:  $P2\text{-real-model } M \sigma z \mathcal{A} = P2\text{-real-model}' M \sigma z \mathcal{A}$

*<proof>*

**lemma** *P2-ideal-view-unfold*:  $\text{mal-def.ideal-view-2 } (x0, x1) \sigma z (P2\text{-S1}, P2\text{-S2}) \mathcal{A}$

= *P2-ideal-model* ( $x0, x1$ )  $\sigma z \mathcal{A}$   
 ⟨*proof*⟩

**end**

**locale** *ot* = *ot-base* + *cyclic-group*  $\mathcal{G}$   
**begin**

**lemma** *P1-assert-correct1*:

**shows**  $((\mathbf{g} [\uparrow] (\alpha 0 :: \text{nat})) [\uparrow] (r :: \text{nat}) \otimes \mathbf{g} \otimes \text{inv} ((\mathbf{g} [\uparrow] (\alpha 1 :: \text{nat})) [\uparrow] r \otimes \mathbf{g}))$   
 $= (\mathbf{g} [\uparrow] \alpha 0 \otimes \text{inv} (\mathbf{g} [\uparrow] \alpha 1)) [\uparrow] r$   
 (**is** *?lhs* = *?rhs*)  
 ⟨*proof*⟩

**lemma** *P1-assert-correct2*:

**shows**  $(\mathbf{g} [\uparrow] (\alpha 0 :: \text{nat})) [\uparrow] (r :: \text{nat}) \otimes \text{inv} ((\mathbf{g} [\uparrow] (\alpha 1 :: \text{nat})) [\uparrow] r) = (\mathbf{g} [\uparrow] \alpha 0$   
 $\otimes \text{inv} (\mathbf{g} [\uparrow] \alpha 1)) [\uparrow] r$   
 (**is** *?lhs* = *?rhs*)  
 ⟨*proof*⟩

**sublocale** *ddh*: *ddh-ext*  
 ⟨*proof*⟩

**lemma** *P1-real-ddh0-σ-false*:

**assumes**  $\sigma = \text{False}$   
**shows**  $((P1\text{-real-model } M \sigma z \mathcal{A}) \gg (\lambda \text{ view. } D \text{ view})) = (\text{ddh.DDH0 } (P1\text{-DDH-mal-adv-}\sigma\text{-false } M z \mathcal{A} D))$   
**including** *monad-normalisation*  
 ⟨*proof*⟩

**lemma** *P1-ideal-ddh1-σ-false*:

**assumes**  $\sigma = \text{False}$   
**shows**  $((P1\text{-ideal-model } M \sigma z \mathcal{A}) \gg (\lambda \text{ view. } D \text{ view})) = (\text{ddh.DDH1 } (P1\text{-DDH-mal-adv-}\sigma\text{-false } M z \mathcal{A} D))$   
**including** *monad-normalisation*  
 ⟨*proof*⟩

**lemma** *P1-real-ddh1-σ-true*:

**assumes**  $\sigma = \text{True}$   
**shows**  $((P1\text{-real-model } M \sigma z \mathcal{A}) \gg (\lambda \text{ view. } D \text{ view})) = (\text{ddh.DDH1 } (P1\text{-DDH-mal-adv-}\sigma\text{-true } M z \mathcal{A} D))$   
**including** *monad-normalisation*  
 ⟨*proof*⟩

**lemma** *P1-ideal-ddh0-σ-true*:

**assumes**  $\sigma = \text{True}$   
**shows**  $((P1\text{-ideal-model } M \sigma z \mathcal{A}) \gg (\lambda \text{ view. } D \text{ view})) = (\text{ddh.DDH0 } (P1\text{-DDH-mal-adv-}\sigma\text{-true } M z \mathcal{A} D))$   
**including** *monad-normalisation*

*<proof>*

**lemma** *P1-real-ideal-DDH-advantage-false:*

**assumes**  $\sigma = \text{False}$

**shows**  $\text{mal-def.adv-P1 } M \ \sigma \ z \ (P1-S1, P1-S2) \ \mathcal{A} \ D = \text{ddh.DDH-advantage}$   
 $(P1-DDH-mal-adv-\sigma\text{-false } M \ z \ \mathcal{A} \ D)$

*<proof>*

**lemma** *P1-real-ideal-DDH-advantage-false-bound:*

**assumes**  $\sigma = \text{False}$

**shows**  $\text{mal-def.adv-P1 } M \ \sigma \ z \ (P1-S1, P1-S2) \ \mathcal{A} \ D$   
 $\leq \text{ddh.advantage } (P1-DDH-mal-adv-\sigma\text{-false } M \ z \ \mathcal{A} \ D)$   
 $+ \text{ddh.advantage } (\text{ddh.DDH-A}' (P1-DDH-mal-adv-\sigma\text{-false } M \ z \ \mathcal{A} \ D))$

*<proof>*

**lemma** *P1-real-ideal-DDH-advantage-true:*

**assumes**  $\sigma = \text{True}$

**shows**  $\text{mal-def.adv-P1 } M \ \sigma \ z \ (P1-S1, P1-S2) \ \mathcal{A} \ D = \text{ddh.DDH-advantage}$   
 $(P1-DDH-mal-adv-\sigma\text{-true } M \ z \ \mathcal{A} \ D)$

*<proof>*

**lemma** *P1-real-ideal-DDH-advantage-true-bound:*

**assumes**  $\sigma = \text{True}$

**shows**  $\text{mal-def.adv-P1 } M \ \sigma \ z \ (P1-S1, P1-S2) \ \mathcal{A} \ D$   
 $\leq \text{ddh.advantage } (P1-DDH-mal-adv-\sigma\text{-true } M \ z \ \mathcal{A} \ D)$   
 $+ \text{ddh.advantage } (\text{ddh.DDH-A}' (P1-DDH-mal-adv-\sigma\text{-true } M \ z \ \mathcal{A} \ D))$

*<proof>*

**lemma** *P2-output-rewrite:*

**assumes**  $s < \text{order } \mathcal{G}$

**shows**  $(\mathbf{g} [\wedge] (r * u1 + v1), \mathbf{g} [\wedge] (r * \alpha * u1 + v1 * \alpha) \otimes \text{inv } \mathbf{g} [\wedge] u1)$   
 $= (\mathbf{g} [\wedge] (r * ((s + u1) \text{ mod } \text{order } \mathcal{G}) + (r * \text{order } \mathcal{G} - r * s + v1) \text{ mod } \text{order } \mathcal{G}),$

$\mathbf{g} [\wedge] (r * \alpha * ((s + u1) \text{ mod } \text{order } \mathcal{G}) + (r * \text{order } \mathcal{G} - r * s + v1)$   
 $\text{ mod } \text{order } \mathcal{G} * \alpha)$

$\otimes \text{inv } \mathbf{g} [\wedge] ((s + u1) \text{ mod } \text{order } \mathcal{G} + (\text{order } \mathcal{G} - s)))$

*<proof>*

**lemma** *P2-inv-g-rewrite:*

**assumes**  $s < \text{order } \mathcal{G}$

**shows**  $(\text{inv } \mathbf{g} [\wedge] (u1' + (\text{order } \mathcal{G} - s))) = \mathbf{g} [\wedge] s \otimes \text{inv } (\mathbf{g} [\wedge] u1')$

*<proof>*

**lemma** *P2-inv-g-s-rewrite:*

**assumes**  $s < \text{order } \mathcal{G}$

**shows**  $\mathbf{g} [\wedge] ((r::\text{nat}) * \alpha * u1 + v1 * \alpha) \otimes \text{inv } \mathbf{g} [\wedge] (u1 + (\text{order } \mathcal{G} - s)) =$

$\mathbf{g} [\uparrow] (r * \alpha * u1 + v1 * \alpha) \otimes \mathbf{g} [\uparrow] s \otimes \text{inv } \mathbf{g} [\uparrow] u1$   
 ⟨proof⟩

**lemma** *P2-e0-rewrite:*

**assumes**  $s < \text{order } \mathcal{G}$   
**shows**  $(\mathbf{g} [\uparrow] (r * x + xa), \mathbf{g} [\uparrow] (r * \alpha * x + xa * \alpha) \otimes \mathbf{g} [\uparrow] x) =$   
 $(\mathbf{g} [\uparrow] (r * ((\text{order } \mathcal{G} - s + x) \text{ mod } \text{order } \mathcal{G}) + (r * s + xa) \text{ mod } \text{order } \mathcal{G}),$   
 $\mathbf{g} [\uparrow] (r * \alpha * ((\text{order } \mathcal{G} - s + x) \text{ mod } \text{order } \mathcal{G}) + (r * s + xa) \text{ mod } \text{order } \mathcal{G} * \alpha)$   
 $\otimes \mathbf{g} [\uparrow] ((\text{order } \mathcal{G} - s + x) \text{ mod } \text{order } \mathcal{G} + s))$   
 ⟨proof⟩

**lemma** *P2-case-l-new-1-gt-e0-rewrite:*

**assumes**  $s < \text{order } \mathcal{G}$   
**shows**  $(\mathbf{g} [\uparrow] (r * ((\text{order } \mathcal{G} * \text{order } \mathcal{G} - s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G}))) + x) \text{ mod } \text{order } \mathcal{G})$   
 $+ (r * s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G})) + xa) \text{ mod } \text{order } \mathcal{G}),$   
 $\mathbf{g} [\uparrow] (r * \alpha * ((\text{order } \mathcal{G} * \text{order } \mathcal{G} - s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G}))) + x) \text{ mod } \text{order } \mathcal{G})$   
 $+ (r * s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G})) + xa) \text{ mod } \text{order } \mathcal{G} * \alpha) \otimes$   
 $\mathbf{g} [\uparrow] (t * ((\text{order } \mathcal{G} * \text{order } \mathcal{G} - s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G}))) + x) \text{ mod } \text{order } \mathcal{G})$   
 $+ s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G})))) = (\mathbf{g} [\uparrow] (r * x + xa), \mathbf{g} [\uparrow] (r * \alpha * x + xa * \alpha) \otimes \mathbf{g} [\uparrow] (t * x))$   
 ⟨proof⟩

**lemma** *P2-case-l-neq-1-gt-x0-rewrite:*

**assumes**  $t < \text{order } \mathcal{G}$   
**and**  $t \neq 0$   
**shows**  $\mathbf{g} [\uparrow] (t * (u0 + (s * (\text{nat } ((\text{fst } (\text{bezw } t (\text{order } \mathcal{G}))) \text{ mod } \text{order } \mathcal{G})))) = \mathbf{g} [\uparrow] (t * u0) \otimes \mathbf{g} [\uparrow] s$   
 ⟨proof⟩

Now we show the two end definitions are equal when the input for l (in the ideal model, the second input) is the one constructed by the simulator

**lemma** *P2-ideal-real-end-eq:*

**assumes**  $b0\text{-inv-}b1: b0 \otimes \text{inv } b1 = (h0 \otimes \text{inv } h1) [\uparrow] r$   
**and**  $\text{assert-in-carrier}: h0 \in \text{carrier } \mathcal{G} \wedge h1 \in \text{carrier } \mathcal{G} \wedge b0 \in \text{carrier } \mathcal{G} \wedge b1 \in \text{carrier } \mathcal{G}$   
**and**  $x1\text{-in-carrier}: x1 \in \text{carrier } \mathcal{G}$   
**and**  $x0\text{-in-carrier}: x0 \in \text{carrier } \mathcal{G}$   
**shows**  $P2\text{-ideal-model-end } (x0, x1) (b0 \otimes (\text{inv } (h0 [\uparrow] r))) ((h0, h1, \mathbf{g} [\uparrow] (r::\text{nat}), b0, b1), s')$   
 $\mathcal{A}3 = P2\text{-real-model-end } (x0, x1) ((h0, h1, \mathbf{g} [\uparrow] (r::\text{nat}), b0, b1), s')$   
**including** *monad-normalisation*  
 ⟨proof⟩

**lemma** *P2-ideal-real-eq*:  
**assumes** *x1-in-carrier*:  $x1 \in \text{carrier } \mathcal{G}$   
**and** *x0-in-carrier*:  $x0 \in \text{carrier } \mathcal{G}$   
**shows**  $P2\text{-real-model } (x0,x1) \sigma z \mathcal{A} = P2\text{-ideal-model } (x0,x1) \sigma z \mathcal{A}$   
 $\langle \text{proof} \rangle$

**lemma** *malicious-sec-P2*:  
**assumes** *x1-in-carrier*:  $x1 \in \text{carrier } \mathcal{G}$   
**and** *x0-in-carrier*:  $x0 \in \text{carrier } \mathcal{G}$   
**shows**  $\text{mal-def.perfect-sec-P2 } (x0,x1) \sigma z (P2\text{-S1}, P2\text{-S2}) \mathcal{A}$   
 $\langle \text{proof} \rangle$

**lemma** *correct*:  
**assumes**  $x0 \in \text{carrier } \mathcal{G}$   
**and**  $x1 \in \text{carrier } \mathcal{G}$   
**shows**  $\text{funct-OT-12 } (x0, x1) \sigma = \text{protocol-ot } (x0,x1) \sigma$   
 $\langle \text{proof} \rangle$

**lemma** *correctness*:  
**assumes**  $x0 \in \text{carrier } \mathcal{G}$   
**and**  $x1 \in \text{carrier } \mathcal{G}$   
**shows**  $\text{mal-def.correct } (x0,x1) \sigma$   
 $\langle \text{proof} \rangle$

**end**

**locale** *OT-asymp* =  
**fixes**  $\mathcal{G} :: \text{nat} \Rightarrow \text{'grp cyclic-group}$   
**assumes**  $\text{ot}: \bigwedge \eta. \text{ot } (\mathcal{G} \eta)$   
**begin**

**sublocale**  $\text{ot } \mathcal{G} \text{ n for } n \langle \text{proof} \rangle$

**lemma** *correctness-asym*:  
**assumes**  $x0 \in \text{carrier } (\mathcal{G} \text{ n})$   
**and**  $x1 \in \text{carrier } (\mathcal{G} \text{ n})$   
**shows**  $\text{mal-def.correct } n (x0,x1) \sigma$   
 $\langle \text{proof} \rangle$

**lemma** *P1-security-asym*:  
*negligible*  $(\lambda n. \text{mal-def.adv-P1 } n M \sigma z (P1\text{-S1 } n, P1\text{-S2}) \mathcal{A} D)$   
**if** *neg1*: *negligible*  $(\lambda n. \text{ddh.advantage } n (P1\text{-DDH-mal-adv-}\sigma\text{-true } n M z \mathcal{A} D))$   
**and** *neg2*: *negligible*  $(\lambda n. \text{ddh.advantage } n (\text{ddh.DDH-A}' n (P1\text{-DDH-mal-adv-}\sigma\text{-true } n M z \mathcal{A} D)))$   
**and** *neg3*: *negligible*  $(\lambda n. \text{ddh.advantage } n (P1\text{-DDH-mal-adv-}\sigma\text{-false } n M z \mathcal{A} D))$   
**and** *neg4*: *negligible*  $(\lambda n. \text{ddh.advantage } n (\text{ddh.DDH-A}' n (P1\text{-DDH-mal-adv-}\sigma\text{-false } n M z \mathcal{A} D)))$

$n M z A D)))$   
 $\langle proof \rangle$

**lemma** *P2-security-asm:*

**assumes** *x1-in-carrier*:  $x1 \in carrier (\mathcal{G} n)$

**and** *x0-in-carrier*:  $x0 \in carrier (\mathcal{G} n)$

**shows** *mal-def.perfect-sec-P2*  $n (x0, x1) \sigma z (P2-S1 n, P2-S2 n) \mathcal{A}$

$\langle proof \rangle$

**end**

**end**

## References

- [1] D. A. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *IACR Cryptology ePrint Archive*, 2017:753, 2017.
- [2] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [3] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.
- [4] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.
- [5] Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.
- [6] A. Lochbihler. CryptHOL. *Archive of Formal Proofs*, 2017. <http://isa-afp.org/entries/CryptHOL.shtml>, Formal proof development.
- [7] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457. ACM/SIAM, 2001.
- [8] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.