

## 1 Overture SYNTAX AND SEMANTICS

$v \in \mathbb{F}_p$ ,  $w \in \text{String}$ ,  $\iota \in \text{Clients} \subset \mathbb{N}$

$\varepsilon ::= r[w] \mid s[w] \mid m[w] \mid p[w] \mid v \mid \varepsilon - \varepsilon \mid \varepsilon + \varepsilon \mid \varepsilon * \varepsilon$  *expressions*

$x ::= r[w]@_\iota \mid s[w]@_\iota \mid m[w]@_\iota \mid p[w] \mid \text{out}@_\iota$  *variables*

$\pi ::= m[w]@_\iota := \varepsilon@_\iota \mid p[w] := e@_\iota \mid \text{out}@_\iota := \varepsilon@_\iota \mid \pi; \pi$  *protocols*

$$\begin{aligned} \llbracket \sigma, v \rrbracket_\iota &= v \\ \llbracket \sigma, \varepsilon_1 + \varepsilon_2 \rrbracket_\iota &= \llbracket \llbracket \sigma, \varepsilon_1 \rrbracket_\iota + \llbracket \sigma, \varepsilon_2 \rrbracket_\iota \rrbracket_\iota \\ \llbracket \sigma, \varepsilon_1 - \varepsilon_2 \rrbracket_\iota &= \llbracket \llbracket \sigma, \varepsilon_1 \rrbracket_\iota - \llbracket \sigma, \varepsilon_2 \rrbracket_\iota \rrbracket_\iota \\ \llbracket \sigma, \varepsilon_1 * \varepsilon_2 \rrbracket_\iota &= \llbracket \llbracket \sigma, \varepsilon_1 \rrbracket_\iota * \llbracket \sigma, \varepsilon_2 \rrbracket_\iota \rrbracket_\iota \\ \llbracket \sigma, r[w] \rrbracket_\iota &= \sigma(r[w]@_\iota) \\ \llbracket \sigma, s[w] \rrbracket_\iota &= \sigma(s[w]@_\iota) \\ \llbracket \sigma, m[w] \rrbracket_\iota &= \sigma(m[w]@_\iota) \\ \llbracket \sigma, p[w] \rrbracket_\iota &= \sigma(p[w]) \end{aligned}$$

$$\frac{(\sigma, x := \varepsilon@_\iota) \Rightarrow \sigma \{x \mapsto \llbracket \sigma, \varepsilon \rrbracket_\iota\} \quad \frac{(\sigma_1, \pi_1) \Rightarrow \sigma_2 \quad (\sigma_2, \pi_2) \Rightarrow \sigma_3}{(\sigma_1, \pi_1; \pi_2) \Rightarrow \sigma_3}}{(\sigma, x := \varepsilon@_\iota) \Rightarrow \sigma \{x \mapsto \llbracket \sigma, \varepsilon \rrbracket_\iota\}}$$

### 1.1 Overture Adversarial Semantics

$\pi ::= \dots \mid \text{assert}(\varepsilon = \varepsilon)$

$$\begin{aligned} (\sigma, x := \varepsilon@_\iota) &\Rightarrow_{\mathcal{A}} \sigma \{x \mapsto \llbracket \sigma, \varepsilon \rrbracket_\iota\} & \iota \in H \\ (\sigma, x := \varepsilon@_\iota) &\Rightarrow_{\mathcal{A}} \sigma \{x \mapsto \llbracket \text{rewrite}_{\mathcal{A}}(\sigma_C, \varepsilon) \rrbracket_\iota\} & \iota \in C \end{aligned}$$

$$\begin{aligned} (\sigma, \text{assert}(\varepsilon_1 = \varepsilon_2)@_\iota) &\Rightarrow_{\mathcal{A}} \sigma & \text{if } \llbracket \sigma, \varepsilon_1 \rrbracket_\iota = \llbracket \sigma, \varepsilon_2 \rrbracket_\iota \text{ or } \iota \in C \\ (\sigma, \text{assert}(\phi(\varepsilon))@_\iota) &\Rightarrow_{\mathcal{A}} \perp & \text{if } \neg \phi(\sigma, \llbracket \sigma, \varepsilon \rrbracket_\iota) \end{aligned}$$

$$\frac{(\sigma_1, \pi_1) \Rightarrow_{\mathcal{A}} \sigma_2 \quad (\sigma_2, \pi_2) \Rightarrow_{\mathcal{A}} \sigma_3}{(\sigma_1, \pi_1; \pi_2) \Rightarrow_{\mathcal{A}} \sigma_3} \quad \frac{(\sigma_1, \pi_1) \Rightarrow_{\mathcal{A}} \perp}{(\sigma_1, \pi_1; \pi_2) \Rightarrow_{\mathcal{A}} \perp}$$

$$\frac{(\sigma_1, \pi_1) \Rightarrow_{\mathcal{A}} \sigma_2 \quad (\sigma_2, \pi_2) \Rightarrow_{\mathcal{A}} \perp}{(\sigma_1, \pi_1; \pi_2) \Rightarrow_{\mathcal{A}} \perp}$$

## 2 Overture CONSTRAINT TYPING

### 2.1 Constraint Satisfiability Modulo Finite Fields

$$\begin{aligned} \phi &::= x \mid \phi + \phi \mid \phi - \phi \mid \phi * \phi \\ E &::= \phi \equiv \phi \mid E \wedge E \end{aligned}$$

We write  $E_1 \models E_2$  iff every model of  $E_1$  is a model of  $E_2$ . Note that this relation is reflexive and transitive.

$$\begin{aligned} \lfloor x \rfloor &= x & \lfloor \varepsilon_1 + \varepsilon_2@_\iota \rfloor &= \lfloor \varepsilon_2@_\iota \rfloor + \lfloor \varepsilon_1@_\iota \rfloor & \lfloor \varepsilon_1 - \varepsilon_2@_\iota \rfloor &= \lfloor \varepsilon_2@_\iota \rfloor - \lfloor \varepsilon_1@_\iota \rfloor \\ \lfloor \varepsilon_1 * \varepsilon_2@_\iota \rfloor &= \lfloor \varepsilon_2@_\iota \rfloor * \lfloor \varepsilon_1@_\iota \rfloor \end{aligned}$$

$$\lfloor \text{OT}(\varepsilon_1 @_{l_1}, \varepsilon_2, \varepsilon_3) @_{l_2} \rfloor = (\lfloor \varepsilon_1 @_{l_1} \rfloor \wedge \lfloor \varepsilon_3 @_{l_2} \rfloor) \vee (\neg \lfloor \varepsilon_1 @_{l_1} \rfloor \wedge \lfloor \varepsilon_2 @_{l_2} \rfloor)$$

$$\lfloor x := \varepsilon @_l \rfloor = x \equiv \lfloor \varepsilon @_l \rfloor \quad \lfloor \text{assert}(\varepsilon_1 = \varepsilon_2) @_l \rfloor = \lfloor \varepsilon_1 @_l \rfloor \equiv \lfloor \varepsilon_2 @_l \rfloor \quad \lfloor \pi_1; \pi_2 \rfloor = \lfloor \pi_1 \rfloor \wedge \lfloor \pi_2 \rfloor$$

The motivating idea is that we can interpret any protocol  $\pi$  as a set of equality constraints  $\lfloor \pi \rfloor$  and use an SMT solver to verify properties relevant to correctness, confidentiality, and integrity. Further, we can leverage entailment relation is critical for efficiency– we can use annotations to obtain a weakened precondition for relevant properties. That is, given  $\pi$ , program annotations or other cues can be used to find a minimal  $E$  with  $\lfloor \pi \rfloor \models E$  for verifying correctness and security.

### 2.1.1 Example: Correctness of 3-Party Addition.

$$\begin{aligned} m[s1]@2 &:= (s[1] - r[\text{local}] - r[x])@1 \\ m[s1]@3 &:= r[x]@1 \\ m[s2]@1 &:= (s[2] - r[\text{local}] - r[x])@2 \\ m[s2]@3 &:= r[x]@2 \\ m[s3]@1 &:= (s[3] - r[\text{local}] - r[x])@3 \\ m[s3]@2 &:= r[x]@3 \\ p[1] &:= (r[\text{local}] + m[s2] + m[s3])@1 \\ p[2] &:= (m[s1] + r[\text{local}] + m[s3])@2 \\ p[3] &:= (m[s1] + m[s2] + r[\text{local}])@3 \\ \text{out}@1 &:= (p[1] + p[2] + p[3])@1 \\ \text{out}@2 &:= (p[1] + p[2] + p[3])@2 \\ \text{out}@3 &:= (p[1] + p[2] + p[3])@3 \end{aligned}$$

Letting  $\pi$  be this protocol, we can verify correctness as:

$$\lfloor \pi \rfloor \models \text{out}@3 \equiv s[1]@1 + s[2]@2 + s[3]@3$$

## 2.2 Confidentiality Types

$$\begin{aligned} t &::= x \mid c(x, T) \\ T &\in 2^t \\ \Gamma &::= \emptyset \mid \Gamma; x : T \end{aligned}$$

*Definition 2.1.*  $R_1; R_2 = R_1 \cup R_2$  iff  $R_1 \cap R_2 = \emptyset$ .

$$\begin{array}{c} \text{DEFTY} \\ \frac{}{\emptyset, E \vdash \phi : \text{vars}(\phi)} \end{array} \quad \frac{\text{ENCODE} \quad E \models \phi \equiv \phi' \oplus r[w]@_l \quad \oplus \in \{+, -\} \quad R, E \vdash \phi' : T}{R; \{r[w]@_l\}, E \vdash \phi : \{c(r[w]@_l, T)\}} \quad \frac{\text{SEND} \quad R, E \vdash \phi : T}{R, E \vdash x \equiv \phi : (x : T)} \quad \frac{\text{SEQ} \quad R_1, E \vdash \phi_1 : \Gamma_1 \quad R_2, E \vdash \phi_2 : \Gamma_2}{R_1; R_2, E \vdash \phi_1 \wedge \phi_2 : \Gamma_1; \Gamma_2}$$

*Definition 2.2.* Given preprocessing predicate  $E_{pre}$  and protocol  $\pi$  we say  $R, E \vdash E_{pre} \wedge \lfloor \pi \rfloor : \Gamma$  is *valid* iff it is derivable and  $E_{pre} \wedge \lfloor \pi \rfloor \models E$ .

$$\begin{array}{c}
\frac{\iota \in C}{\Gamma, C \vdash_{leak} \Gamma(m[w]@_{\iota})} \quad \frac{\Gamma, C \vdash_{leak} T_1 \cup T_2}{\Gamma, C \vdash_{leak} T_1} \quad \frac{\Gamma, C \vdash_{leak} \{m[w]@_{\iota}\}}{\Gamma, C \vdash_{leak} \Gamma(m[w]@_{\iota})} \\
\\
\frac{\Gamma, C \vdash_{leak} \{r[w]@_{\iota}\} \quad \Gamma, C \vdash_{leak} \{c(r[w]@_{\iota}, T)\}}{\Gamma, C \vdash_{leak} T}
\end{array}$$

**THEOREM 2.3.** *If  $R, E \vdash E_{pre} \wedge [\pi] : \Gamma$  is valid and there exists no  $H, C$  and  $s[w]@_{\iota}$  for  $\iota \in H$  with  $\Gamma, C \vdash_{leak} \{s[w]@_{\iota}\}$ , then  $\pi$  satisfies gradual release.*

### 2.2.1 Examples.

$m[s1]@2 := (s[1] - r[local] - r[x])@1$   
 $m[s1]@3 := r[x]@1$

//  $m[s1]@2 : \{ c(r[x]@1, \{ c(r[local]@1, \{ s[1]@1 \} ) \} )$

//  $m[s1]@3 : \{ r[x]@1 \}$

$m[x]@1 := s2(s[x], -r[x], r[x])@2$

//  $m[x]@1 == s[x]@2 + -r[x]@2$

//  $m[x]@1 : \{ c(r[x]@2, \{ s[x]@2 \} ) \}$

$m[y]@1 := 0T(s[y]@1, -r[y], r[y])@2$

//  $m[y]@1 == s[y]@1 + -r[y]@2$

//  $m[y]@1 : \{ c(r[y]@2, \{ s[y]@1 \} ) \}$

## 2.3 Integrity Types

$\zeta ::= \text{High} \mid \text{Low}$

$\Gamma ::= \emptyset \mid \Gamma; x : \zeta$

VALUE	SECRET	RANDO	MESG
$\Gamma, E \vdash v : \text{High}$	$\Gamma, E \vdash s[w] : \mathcal{L}(\iota)$	$\Gamma, E \vdash r[w] : \mathcal{L}(\iota)$	$\Gamma, E \vdash m[w] : \Gamma(m[w]@_{\iota})$

PUBM	BINOP
$\Gamma, E \vdash p[w] : \Gamma(p[w])$	$\frac{\Gamma, E \vdash \varepsilon_1 : \zeta \quad \Gamma, E \vdash \varepsilon_2 : \zeta \quad \oplus \in \{+, -, *\}}{\Gamma, E \vdash \varepsilon_1 \oplus \varepsilon_2 : \zeta}$

INTEGRITYWEAKEN
$\frac{\Gamma, E \vdash \varepsilon : \zeta_1 \quad \zeta_1 \leq \zeta_2}{\Gamma, E \vdash \varepsilon : \zeta_2}$

SEND	SEQ
$\frac{\Gamma, E \vdash \varepsilon : \mathcal{L}(\iota)}{\Gamma, E \vdash x := \varepsilon@_{\iota} : \Gamma; x : \mathcal{L}(\iota)}$	$\frac{\Gamma_1, E \vdash \pi_1 : \Gamma_2 \quad \Gamma_2, E \vdash \pi_2 : \Gamma_3}{\Gamma_1, E \vdash \pi_1; \pi_2 : \Gamma_3}$

MAC
$\frac{E \models [\text{assert}(\psi_{BDOZ}(w))@_{\iota}]}{\Gamma, E \vdash \text{assert}(\psi_{BDOZ}(w))@_{\iota} : \Gamma; m[ws]@_{\iota} : \text{High}}$

$$\psi_{BDOZ}(w) \triangleq m[wm] = m[wk] + (m[\text{delta}] * m[ws])$$

### 3 COMPOSITIONAL TYPE VERIFICATION IN *Prelude*

Note the redefinition of  $x$  impacts the definition of  $T$ ,  $\Gamma$ ,  $\phi$ , and  $E$ .

$$\begin{aligned} x &::= r[e]@e \mid s[e]@e \mid m[e]@e \mid p[e] \mid \text{out}@e \\ \ell \in \text{Field}, y \in \text{EVar}, f \in \text{FName} \\ e &::= v \mid r[e] \mid s[e] \mid m[e] \mid p[e] \mid e \text{ binop } e \mid \text{let } y = e \text{ in } e \mid \\ &\quad f(e, \dots, e) \mid \{\ell = e; \dots; \ell = e\} \mid e.\ell \\ c &::= m[e]@e := e@e \mid p[e] := e@e \mid \text{out}@e := e@e \mid \text{assert}(e = e)@e \mid \\ &\quad f(e, \dots, e) \mid c; c \mid m[e]@e \text{ as } \phi \\ \text{binop} &::= + \mid - \mid * \mid ++ \\ v &::= w \mid \iota \mid \varepsilon \mid \{\ell = v; \dots; \ell = v\} \\ fn &::= f(y, \dots, y)\{e\} \mid f(y, \dots, y)\{c\} \end{aligned}$$

$$R \Vdash x : (\emptyset, \{x\}) \quad \frac{R \Vdash \phi : (R_1, T) \quad r[w]@_\iota \notin R \quad \oplus \in \{+, -\}}{R_1 \Vdash \phi \oplus r[w]@_\iota : (R_1 \cup \{r[w]@_\iota\}, \{c(r[w]@_\iota, T)\})}$$

$$\frac{R \Vdash \phi_1 : (R_1, T_1) \quad R \Vdash \phi_2 : (R_2, T_2) \quad \oplus \in \{+, -, *\}}{R_1 \Vdash \phi_1 \oplus \phi_2 : (R_1; R_2, T_1 \cup T_2)}$$

$$\frac{e[v/y] \Rightarrow v'}{\text{let } y = v \text{ in } e \Rightarrow v'}$$

$$\frac{C(f) = y_1, \dots, y_n, e \quad e_1 \Rightarrow v_1 \cdots e_n \Rightarrow v_n \quad e[v_1/y_1] \cdots [v_n/y_n] \Rightarrow v}{f(e_1, \dots, e_n) \Rightarrow v}$$

$$\frac{e_1 \Rightarrow v_1 \cdots e_n \Rightarrow v_n}{\{\ell_1 = e_1; \dots; \ell_n = e_n\} \Rightarrow \{\ell_1 = v_1; \dots; \ell_n = v_n\}} \quad \frac{e \Rightarrow \{\dots; \ell = v; \dots\}}{e.\ell \Rightarrow v}$$

$$\frac{e_1 \Rightarrow w_1 \quad e_2 \Rightarrow w_2}{e_1 ++ e_2 \Rightarrow w_1 w_2} \quad \frac{e \Rightarrow w}{m[e] \Rightarrow w} \quad \frac{e \Rightarrow w}{m[e]@_\iota \Rightarrow m[w]@_\iota} \quad \frac{e_1 \Rightarrow \varepsilon_1 \quad e_2 \Rightarrow \varepsilon_2}{e_1 + e_2 \Rightarrow \varepsilon_1 + \varepsilon_2}$$

$$\frac{e_1 \Rightarrow x_1 \quad e_2 \Rightarrow x_2}{e_1 * e_2 \Rightarrow x_1 * x_2} \quad \frac{e_1 \Rightarrow \phi_1 \quad e_2 \Rightarrow \phi_2}{e_1 \equiv e_2 \Rightarrow \phi_1 \equiv \phi_2} \quad \frac{e_1 \Rightarrow E_1 \quad e_2 \Rightarrow E_2}{e_1 \wedge e_2 \Rightarrow E_1 \wedge E_2}$$

$$\frac{\text{MMSG} \quad e_1 \Rightarrow \varepsilon \quad e_2 \Rightarrow \iota \quad R_1 \Vdash \lfloor \varepsilon @ \iota \rfloor : (R_2, T)}{R_1 \vdash x := e_1 @ e_2 : \{E\} \ x : T, R_1; R_2 \ \{E \wedge x \equiv \lfloor \varepsilon @ \iota \rfloor\}}$$

$$\frac{\text{ENCODE} \quad e_1 \Rightarrow w \quad e_2 \Rightarrow \iota \quad E \models \lfloor \varepsilon @ \iota \rfloor \equiv \phi \quad R_1 \Vdash \phi : (R_2, T)}{R_1 \vdash m[e_1] @ e_2 \text{ as } \phi : \{E\} \ m[w] @ \iota : T, R_1; R_2 \ \{E \wedge m[w] @ \iota \equiv \phi\}}$$

$$\frac{\text{APP} \quad \begin{array}{l} \text{sig}(f) = \Pi x_1, \dots, x_n. \{E_1\} \ \Gamma, R \ \{E_2\} \\ e_1 \Rightarrow v_1 \ \dots \ e_n \Rightarrow v_n \quad \rho = [v_1/x_1] \ \dots \ [v_n/x_n] \quad E \models \rho(E_1) \end{array}}{R_1 \vdash f(e_1, \dots, e_n) : \{E\} \ \rho(\Gamma), R_1; \rho(R) \ \{E \wedge \rho(E_2)\}}$$

$$\frac{\text{SEQ} \quad \begin{array}{l} R_1 \vdash \pi_1 : \{E_1\} \ \Gamma_2, R_2 \ \{E_2\} \quad R_1 \vdash \pi_2 : \{E_2\} \ \Gamma_3, R_3 \ \{E_3\} \end{array}}{R_1 \vdash \pi_1; \pi_2 : \{E_1\} \ \Gamma_2; \Gamma_3, R_2; R_3 \ \{E_3\}}$$

$$\frac{\text{SIG} \quad \begin{array}{l} C(f) = x_1, \dots, x_n, c \\ \rho = [v_1/x_1] \ \dots \ [v_n/x_n] \quad \emptyset \vdash \rho(c) : \{\rho(E_1)\} \ \rho(\Gamma), \rho(R) \ \{E\} \quad E \models \rho(E_2) \end{array}}{f : \Pi x_1, \dots, x_n. \{E_1\} \ \Gamma, R \ \{E_2\}}$$

*Definition 3.1.*  $\text{sig}$  is verified iff  $f : \text{sig}(f)$  is valid for all  $f \in \text{dom}(\text{sig})$ .

The following theorem holds for protocols with default preprocessing.

**THEOREM 3.2.** *If  $\text{sig}$  is verified and  $\emptyset \vdash e : \{\emptyset\} \ \Gamma, R \ \{E\}$  then  $e \Rightarrow \pi$  and  $R, E \vdash \pi : \Gamma$  is valid.*

### 3.1 Confidentiality Examples

```

andtableygc(g,x,y)
{
  let table = (~r[g],~r[g],~r[g],r[g])
  in permute4(r[x],r[y],table)
}

m[x]@1 := s2(s[x],r[x],~r[x])@2;
m[x]@1 as s[x]@2 xor r[x]@2;

// m[x]@1 : { c(r[x]@2, { s[x]@2 }) }

m[y]@1 := OT(s[y]@1,r[y],~r[y])@2;
m[y]@1 as s[y]@1 xor r[y]@2;

// m[y]@1 : { c(r[y]@2, { s[y]@1 }) }

m[ag]@1 := OT4(m[x]@1, m[y]@1, andtable(ag,r[x],r[y]))@2;
m[ag]@1 as ~(r[x]@2 = m[x]@1) and (r[y]@2 = m[y]@1) xor r[ag]@2;

```

```

197 // m[ag]@1 : { c(r[ag]@2, {r[x]@2, r[y]@2, m[x]@1, m[y]@1} }
198
199 p[o] := OT2(m[ag]@1, perm2(r[ag],(false,true)))@2
200
201 // p[o] : { c(r[ag]@2, {r[x]@2, r[y]@2, m[x]@1, m[y]@1}), r[ag]@2 }
202
203 out@1 := p[o]@1
204
205 // out@1 == s[x] and s[y]
206
207 encodegmw(in, i1, i2) {
208   m[in]@i2 := (s[in] xor r[in])@i1;
209   m[in]@i1 := r[in]@i1
210 }
211
212 andtablegmw(x, y, z) {
213   let r11 = r[z] xor (m[x] xor true) and (m[y] xor true) in
214   let r10 = r[z] xor (m[x] xor true) and (m[y] xor false) in
215   let r01 = r[z] xor (m[x] xor false) and (m[y] xor true) in
216   let r00 = r[z] xor (m[x] xor false) and (m[y] xor false) in
217   { row1 = r11; row2 = r10; row3 = r01; row4 = r00 }
218 }
219
220 andgmw(z, x, y) {
221   let table = andtablegmw(x,y,z) in
222   m[z]@2 := OT4(m[x],m[y],table,2,1);
223   m[z]@2 as ~((m[x]@1 xor m[x]@2) and (m[y]@1 xor m[y]@2)) xor r[z]@1);
224   m[z]@1 := r[z]@1
225 }
226
227 // and gate correctness postcondition
228 { } andgmw { m[z]@1 xor m[z]@2 == (m[x]@1 xor m[x]@2) and (m[y]@1 xor m[y]@2) }
229
230 // and gate type
231 andgmw :
232   Pi z,x,y .
233   { }
234   { { r[z]@1 },
235     (m[z]@1 : { r[z]@1 }; m[z]@2 : {c(r[z]@1, { m[x]@1, m[x]@2, m[y]@1, m[y]@2 })} ),
236     m[z]@1 xor m[z]@2 == (m[x]@1 xor m[x]@2) and (m[y]@1 xor m[y]@2)}
237
238 xorgmw(z, x, y) {
239   m[z]@1 := (m[x] xor m[y])@1; m[z]@2 := (m[x] xor m[y])@2;
240 }
241
242 decodegmw(z) {
243   p["1"] := m[z]@1; p["2"] := m[z]@2;
244   out@1 := (p["1"] xor p["2"])@1;
245

```

```

246     out@2 := (p["1"] xor p["2"]>@2
247 }
248
249 prot() {
250     encodegmw("x",2,1);
251     encodegmw("y",2,1);
252     encodegmw("z",1,2);
253     andgmw("g1","x","z");
254     xorgmw("g2","g1","y");
255     decodegmw("g2")
256 }
257
258 {} prot { out@1 == (s["x"]@1 and s["z"]@2) xor s["y"]@1 }
259

```

### 3.2 Integrity Examples

```

260
261 secopen(w1,w2,w3,i1,i2) {
262     pre(m[w1++"m"]@i2 == m[w1++"k"]@i1 + (m["delta"]@i1 * m[w1++"s"]@i2 /\
263         m[w1++"m"]@i2 == m[w1++"k"]@i1 + (m["delta"]@i1 * m[w1++"s"]@i2));
264     let locsum = macsum(macshare(w1), macshare(w2)) in
265     m[w3++"s"]@i1 := (locsum.share)@i2;
266     m[w3++"m"]@i1 := (locsum.mac)@i2;
267     auth(m[w3++"s"],m[w3++"m"],mack(w1) + mack(w2),i1);
268     m[w3]@i1 := (m[w3++"s"] + (locsum.share))@i1
269 }
270
271
272 _open(x,i1,i2){
273     m[x++"exts"]@i1 := m[x++"s"]@i2;
274     m[x++"extm"]@i1 := m[x++"m"]@i2;
275     assert(m[x++"extm"] == m[x++"k"] + (m["delta"] * m[x++"exts"]));
276     m[x]@i1 := (m[x++"exts"] + m[x++"s"]>@i2
277 }`
278
279 _sum(z, x, y,i1,i2) {
280     pre(m[x++"m"]@i2 == m[x++"k"]@i1 + (m["delta"]@i1 * m[x++"s"]@i2 /\
281         m[y++"m"]@i2 == m[y++"k"]@i1 + (m["delta"]@i1 * m[y++"s"]@i2));
282     m[z++"s"]@i2 := (m[x++"s"] + m[y++"s"]>@i2;
283     m[z++"m"]@i2 := (m[x++"m"] + m[y++"m"]>@i2;
284     m[z++"k"]@i1 := (m[x++"k"] + m[y++"k"]>@i1;
285     post(m[z++"m"]@i2 == m[z++"k"]@i1 + (m["delta"]@i1 * m[z++"s"]@i2)
286 }
287
288 sum(z,x,y) { _sum(z,x,y,1,2);_sum(z,x,y,2,1) }
289
290 open(x) { _open(x,1,2); _open(x,2,1) }
291
292
293 sum("a", "x", "d");
294

```

```

295 open("d");
296 sum("b", "y", "e");
297 open("e");
298 let xys =
299     macsum(macctimes(macshare("b"), m["d"]),
300           macsum(macctimes(macshare("a"), m["e"]),
301                 macshare("c")))
302 let xyk = mack("b") * m["d"] + mack("a") * m["e"] + mack("c")
303
304 secopen("a", "x", "d", 1, 2);
305 secopen("a", "x", "d", 2, 1);
306 secopen("b", "y", "e", 1, 2);
307 secopen("b", "y", "e", 2, 1);
308 let xys =
309     macsum(macctimes(macshare("b"), m["d"]),
310           macsum(macctimes(macshare("a"), m["e"]),
311                 macshare("c")))
312 in
313 let xyk = mack("b") * m["d"] + mack("d") * m["d"] + mack("c")
314 in
315 secreveal(xys, xyk, "1", 1, 2);
316 secreveal(maccsum(xys, m["d"] * m["e"]),
317           xyk - m["d"] * m["e"],
318           "2", 2, 1);
319 out@1 := (p[1] + p[2])@1;
320 out@2 := (p[1] + p[2])@2;
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343

```