

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 π 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 π , 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 π 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 $R_1 \cap R_2 = \emptyset$.

$$\begin{array}{c} \text{DEFTY} \\ \hline \emptyset, E \vdash \phi : \text{vars}(\phi) \end{array} \quad \frac{\text{ENCODE} \quad E \models \phi \equiv \phi' \oplus r[w]@_l \quad \oplus \in \{+, -\} \quad R, E \vdash \phi' : T}{R; \{r[w]@_l\}, E \vdash \phi : \{c(r[w]@_l, T)\}}$$

$$\begin{array}{c} \text{SEND} \\ \hline R, E \vdash \phi : T \\ \hline R, E \vdash x \equiv \phi : (x : T) \end{array} \quad \frac{\text{SEQ} \quad R_1, E \vdash \phi_1 : \Gamma_1 \quad R_2, E \vdash \phi_2 : \Gamma_2}{R_1; R_2, E \vdash \phi_1 \wedge \phi_2 : \Gamma_1; \Gamma_2}$$

Definition 2.2. Given preprocessing predicate E_{pre} and protocol π 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 $R, E \vdash E_{pre} \wedge \lfloor \pi \rfloor : \Gamma$ 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 π 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 := OT(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

$$\begin{array}{l}
\zeta ::= \text{High} \mid \text{Low} \\
\Gamma ::= \emptyset \mid \Gamma; x : \zeta
\end{array}$$

$$\begin{array}{llll}
\text{VALUE} & \text{SECRET} & \text{RANDO} & \text{MESG} \\
\Gamma, E \vdash_{\iota} v : \text{High} & \Gamma, E \vdash_{\iota} s[w] : \iota & \Gamma, E \vdash_{\iota} r[w] : \iota & \Gamma, E \vdash_{\iota} m[w] : \Gamma(m[w]@_{\iota})
\end{array}$$

$$\begin{array}{c}
\text{PUBM} \\
\Gamma, E \vdash_{\iota} p[w] : \Gamma(p[w])
\end{array}
\quad
\frac{\text{BINOP} \quad \Gamma, E \vdash_{\iota} \varepsilon_1 : I_1 \quad \Gamma, E \vdash_{\iota} \varepsilon_2 : I_2 \quad \oplus \in \{+, -, *\}}{\Gamma, E \vdash_{\iota} \varepsilon_1 \oplus \varepsilon_2 : I_1 \sqcap I_2}$$

$$\begin{array}{c}
\text{SEND} \\
\frac{\Gamma, E \vdash_{\iota} \varepsilon : I}{\Gamma, E \vdash x := \varepsilon@_{\iota} : \Gamma; x : \iota \sqcap I}
\end{array}
\quad
\frac{\text{SEQ} \quad \Gamma_1, E \vdash \pi_1 : \Gamma_2 \quad \Gamma_2, E \vdash \pi_2 : \Gamma_3}{\Gamma_1, E \vdash \pi_1; \pi_2 : \Gamma_3}$$

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

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

$$\llbracket \zeta \rrbracket_{H,C} = \zeta \quad \frac{\iota \in H}{\llbracket \iota \rrbracket_{H,C} = \text{High}} \quad \frac{\iota \in C}{\llbracket \iota \rrbracket_{H,C} = \text{Low}} \quad \llbracket I_1 \sqcap I_2 \rrbracket_{H,C} = \llbracket I_1 \rrbracket_{H,C} \wedge \llbracket I_2 \rrbracket_{H,C}$$

Definition 2.4. Given H, C and pre-processing predicate E_{pre} defining M , define $\text{init}(E_{pre}) \triangleq \Gamma$ where for all $m[w]@l \in M$ we have $\Gamma(m[w]@l) = l$.

Definition 2.5. Given pre-processing predicate E_{pre} and protocol π , we say $\text{init}(E_{pre}), E \vdash \pi : \Gamma$ is valid iff it is derivable and $E_{pre} \wedge \lfloor \pi \rfloor \models E$.

THEOREM 2.6. Given pre-processing predicate E_{pre} and protocol π , if $\text{init}(E_{pre}), E \vdash \pi : \Gamma$ is valid and for all H, C and $x \in V_{H \triangleright C}$ we have $\llbracket \Gamma(x) \rrbracket_{H,C} = \text{High}$, then cheating is detectable in π .

3 COMPOSITIONAL TYPE VERIFICATION IN *Prelude*

Note the redefinition of x impacts the definition of T, Γ, ϕ , 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\} \\ R \Vdash x : (\emptyset, \{x\}) &\quad \frac{R \Vdash \phi : (R_1, T) \quad r[w]@l \notin R \quad \oplus \in \{+, -\}}{R_1 \Vdash \phi \oplus r[w]@l : (R_1 \cup \{r[w]@l\}, \{c(r[w]@l, T)\})} \\ &\quad \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)} \\ &\quad \frac{e[v/y] \Rightarrow v'}{\text{let } y = v \text{ in } e \Rightarrow v'} \\ &\quad \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} \\ &\quad \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} \\ &\quad \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]@l \Rightarrow m[w]@l} \quad \frac{e_1 \Rightarrow \varepsilon_1 \quad e_2 \Rightarrow \varepsilon_2}{e_1 + e_2 \Rightarrow \varepsilon_1 + \varepsilon_2} \\ &\quad \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} \end{aligned}$$

$$\begin{array}{c}
\text{148} \\
\text{149} \\
\text{150} \\
\text{151} \\
\text{152} \\
\text{153} \\
\text{154} \\
\text{155} \\
\text{156} \\
\text{157} \\
\text{158} \\
\text{159} \\
\text{160} \\
\text{161} \\
\text{162} \\
\text{163} \\
\text{164} \\
\text{165} \\
\text{166} \\
\text{167} \\
\text{168} \\
\text{169} \\
\text{170}
\end{array}$$

$$\begin{array}{c}
\text{MSG} \\
\frac{e_1 \Rightarrow x \quad e_2 \Rightarrow \varepsilon \quad e_3 \Rightarrow \iota \quad R_1 \Vdash \lfloor \varepsilon @ \iota \rfloor : (R_2, T)}{R_1 \vdash e_1 := e_2 @ e_3 : \{E\} \ x : T, R_1; R_2 \ \{E \wedge x \equiv \lfloor \varepsilon @ \iota \rfloor\}} \\
\\
\text{ENCODE} \\
\frac{e_1 \Rightarrow w \quad e_2 \Rightarrow \iota \quad e_3 \Rightarrow \phi \quad E \models \lfloor \varepsilon @ \iota \rfloor \equiv \phi \quad R_1 \Vdash \phi : (R_2, T)}{R_1 \vdash m[e_1] @ e_2 \text{ as } e_3 : \{E\} \ m[w] @ \iota : T, R_1; R_2 \ \{E \wedge m[w] @ \iota \equiv \phi\}} \\
\\
\text{APP} \\
\frac{\text{sig}(f) = \Pi y_1, \dots, y_n. \{\check{E}_1\} \check{\Gamma}, \check{R} \ \{\check{E}_2\} \quad e_1 \Rightarrow v_1 \cdots e_n \Rightarrow v_n \quad \rho = [v_1/y_1] \cdots [v_n/y_n] \quad \rho(\{\check{E}_1\} \check{\Gamma}, \check{R} \ \{\check{E}_2\}) \Rightarrow \{E_1\} \Gamma, R \ \{E_2\} \quad E \models E_1}{R_1 \vdash f(e_1, \dots, e_n) : \{E\} \ \Gamma, R_1; R \ \{E \wedge E_2\}} \\
\\
\text{SEQ} \\
\frac{R_1 \vdash \pi_1 : \{E_1\} \ \Gamma_2, R_2 \ \{E_2\} \quad R_1 \vdash \pi_2 : \{E_2\} \ \Gamma_3, R_3 \ \{E_3\}}{R_1 \vdash \pi_1; \pi_2 : \{E_1\} \ \Gamma_2; \Gamma_3, R_2; R_3 \ \{E_3\}} \\
\\
\text{SIG} \\
\frac{C(f) = y_1, \dots, y_n, \mathbf{c} \quad \rho = [v_1/y_1] \cdots [v_n/y_n] \quad \rho(\{\check{E}_1\} \check{\Gamma}, \check{R} \ \{\check{E}_2\}) \Rightarrow \{E_1\} \Gamma, R \ \{E_2\} \quad \emptyset \vdash \rho(\mathbf{c}) : \{E_1\} \ \Gamma, R \ \{E\} \quad E \models E_2}{f : \Pi y_1, \dots, y_n. \{\check{E}_1\} \check{\Gamma}, \check{R} \ \{\check{E}_2\}}
\end{array}$$

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

THEOREM 3.2. Given preprocessing predicate E_{pre} , program \mathbf{c} , and verified sig, if $E_{pre} \vdash \mathbf{c} : \{\emptyset\} \ \Gamma, R \ \{E\}$ then $\mathbf{c} \Rightarrow \pi$ and:

- (1) $R, E \vdash E_{pre} \wedge \lfloor \pi \rfloor : \Gamma | 1$ is valid.
- (2) $\text{init}(E_{pre}), E \vdash \pi : \Gamma | 2$ 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;

```

```

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

```

```

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

```

3.2 Integrity Examples

```

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

```

```

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

```