

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*

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

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

1.1 Overture Adversarial Semantics

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

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

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

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

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

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 $R_1 \cap R_2 = \emptyset$.

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

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

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

$$\begin{array}{l}
\varsigma ::= \text{High} \mid \text{Low} \\
\Delta ::= \emptyset \mid \Delta; x : \iota \cdot V
\end{array}$$

VALUE	SECRET	RANDO	MESG	PUBM
$\vdash_{\iota} v : \emptyset$	$\vdash_{\iota} s[w] : \emptyset$	$\vdash_{\iota} r[w] : \emptyset$	$\vdash_{\iota} m[w] : \{m[w]@_{\iota}\}$	$\vdash_{\iota} p[w] : \{p[w]\}$

$$\begin{array}{c}
\text{BINOP} \\
\frac{\vdash_{\iota} \varepsilon_1 : V_1 \quad \vdash_{\iota} \varepsilon_2 : V_2 \quad \oplus \in \{+, -, *\}}{\vdash_{\iota} \varepsilon_1 \oplus \varepsilon_2 : V_1 \cup V_2}
\end{array}$$

$$\begin{array}{c}
\text{SEND} \\
\frac{\vdash_{\iota} \varepsilon : V}{E \vdash x := \varepsilon@_{\iota} : (x : \iota \cdot V)}
\end{array}
\quad
\begin{array}{c}
\text{SEQ} \\
\frac{E \vdash \pi_1 : \Delta_1 \quad E \vdash \pi_2 : \Delta_2}{E \vdash \pi_1; \pi_2 : \Delta_1; \Delta_2}
\end{array}$$

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

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

$$\emptyset \underset{H,C}{\rightsquigarrow} \mathcal{L}_{H,C} \quad \frac{\Delta \underset{H,C}{\rightsquigarrow} \mathcal{L} \quad \iota \in H}{\Delta; x : \iota \cdot V \underset{H,C}{\rightsquigarrow} \mathcal{L}\{x \mapsto \text{High} \wedge (\bigwedge_{x \in V} \mathcal{L}_2(x))\}} \quad \frac{\Delta \underset{H,C}{\rightsquigarrow} \mathcal{L} \quad \iota \in C}{\Delta; x : \iota \cdot V \underset{H,C}{\rightsquigarrow} \mathcal{L}\{x \mapsto \text{Low}\}}$$

Definition 2.4. Given pre-processing predicate E_{pre} and protocol π , we say $E \vdash \pi : \Delta$ is *valid* iff it is derivable and $E_{pre} \wedge \lfloor \pi \rfloor \models E$.

Definition 2.5. Given H, C , define $\mathcal{L}_{H,C}$ such that for all $m[w]@_i$ we have $\mathcal{L}_{H,C}(m[w]@_i) = \text{High}$ if $i \in H$ and Low otherwise.

THEOREM 2.6. Given pre-processing predicate E_{pre} and protocol π with $\text{views}(\pi) = V$, if $E \vdash \pi : \Delta$ is valid and for all H, C with $\Delta \underset{H,C}{\rightsquigarrow} \mathcal{L}$ we have $\mathcal{L}(x) = \text{High}$ for all $x \in V_{H \triangleright C}$, then cheating is detectable in π .

3 COMPOSITIONAL TYPE VERIFICATION IN *Prelude*

3.1 Syntax and Semantics

$\ell \in \text{Field}, y \in \text{EVar}, f \in \text{FName}$

$e ::= v \mid r[e] \mid s[e] \mid m[e] \mid p[e] \mid \text{out} \mid e \text{ binop } e \mid \text{let } y = e \text{ in } e \mid f(e, \dots, e) \mid \{\ell = e; \dots; \ell = e\} \mid e.\ell \mid e@e \mid y$

$c ::= e := e \mid \text{assert}(e = e) \mid f(e, \dots, e) \mid c; c$

$\text{binop} ::= + \mid - \mid * \mid ++$

$v ::= w \mid \iota \mid \varepsilon \mid x \mid \{\ell = v; \dots; \ell = v\}$

$fn ::= f(y, \dots, y)\{e\} \mid f(y, \dots, y)\{c\}$

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

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

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

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

$$\frac{e_1 \Rightarrow x \quad e_2 \Rightarrow \varepsilon @ \iota}{e_1 := e_2 \Rightarrow x := \varepsilon @ \iota} \quad \frac{e_1 \Rightarrow \pi_1 \quad e_2 \Rightarrow \pi_2}{e_1; e_2 \Rightarrow \pi_1; \pi_2} \quad \frac{e_1 \Rightarrow \varepsilon_1 \quad e_2 \Rightarrow \varepsilon_2}{\text{assert}(e_1 = e_2) \Rightarrow \text{assert}(\varepsilon_1 = \varepsilon_2)}$$

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

3.2 Dependent Hoare Type Theory

$$\begin{aligned}
& c ::= \dots \mid m[e]@e \text{ as } \check{\phi} \\
& \check{\phi} ::= e \mid \check{\phi} + \check{\phi} \mid \check{\phi} - \check{\phi} \mid \check{\phi} * \check{\phi} \\
& \check{E} ::= \check{\phi} \equiv \check{\phi} \mid \check{E} \wedge \check{E} \\
& \check{t} ::= e \mid c(e, \check{T}) \\
& \check{T} \in 2^{\check{t}} \\
& \check{\Gamma} ::= \emptyset \mid \check{\Gamma}; e : \check{T} \\
& \check{\Delta} ::= \emptyset \mid \check{\Delta}; e : e \cdot \check{V} \\
& \check{X} \in 2^e \\
\\
& \frac{\check{\phi}_1 \Rightarrow \phi_1 \quad \check{\phi}_2 \Rightarrow \phi_2}{\check{\phi}_1 * \check{\phi}_2 \Rightarrow \phi_1 * \phi_2} \quad \frac{\check{\phi}_1 \Rightarrow \phi_1 \quad \check{\phi}_2 \Rightarrow \phi_2}{\check{\phi}_1 \equiv \check{\phi}_2 \Rightarrow \phi_1 \equiv \phi_2} \quad \frac{\check{E}_1 \Rightarrow E_1 \quad \check{E}_2 \Rightarrow E_2}{\check{E}_1 \wedge \check{E}_2 \Rightarrow E_1 \wedge E_2} \\
& \frac{e \Rightarrow x \quad \check{T} \Rightarrow T}{c(e, \check{T}) \Rightarrow c(x, T)} \quad \frac{\check{t}_1 \Rightarrow t_1 \quad \dots \quad \check{t}_n \Rightarrow t_n}{\{\check{t}_1, \dots, \check{t}_n\} \Rightarrow \{t_1, \dots, t_n\}} \quad \frac{\check{\Gamma} \Rightarrow \Gamma \quad e \Rightarrow x \quad \check{T} \Rightarrow T}{\check{\Gamma}; e : \check{T} \Rightarrow \Gamma; x : T} \\
& \frac{\check{\Delta} \Rightarrow \Delta \quad e_1 \Rightarrow x \quad e_2 \Rightarrow \iota \quad \check{V} \Rightarrow V}{\check{\Delta}; e_1 : e_2 \cdot \check{V} \Rightarrow \Delta; x : \iota \cdot V} \\
& \frac{\check{E}_1 \Rightarrow E_1 \quad \check{\Gamma} \Rightarrow \check{R} \Rightarrow R \quad \check{\Delta} \Rightarrow \Delta \quad \check{E}_2 \Rightarrow E_2}{\{\check{E}_1\} \check{\Gamma}, \check{R} \cdot \check{\Delta} \{\check{E}_2\} \Rightarrow \{E_1\} \Gamma, R \cdot \Delta \{E_2\}} \\
& \vdash x : (\emptyset, \{x\}) \quad \frac{\vdash \phi : (R, T) \quad r[w]@l \notin R \quad \oplus \in \{+, -\}}{\vdash \phi \oplus r[w]@l : (R \cup \{r[w]@l\}, \{c(r[w]@l, T)\})} \\
& \frac{\vdash \phi_1 : (R_1, T_1) \quad \vdash \phi_2 : (R_2, T_2) \quad \oplus \in \{+, -, *\}}{\vdash \phi_1 \oplus \phi_2 : (R_1; R_2, T_1 \cup T_2)} \\
& \text{MESG} \\
& \frac{e_1 \Rightarrow x \quad e_2 \Rightarrow \varepsilon \quad e_3 \Rightarrow \iota \quad \vdash \lfloor \varepsilon @ l \rfloor : (R_2, T) \quad \vdash_l \varepsilon : V}{\vdash e_1 := e_2 @ e_3 : \{E\} (x : T), R_1; R_2 \cdot (x : \iota \cdot V) \{E \wedge x \equiv \lfloor \varepsilon @ l \rfloor\}} \\
& \text{ENCODE} \\
& \frac{e_1 \Rightarrow w \quad e_2 \Rightarrow \iota \quad \check{\phi} \Rightarrow \phi \quad E \models \lfloor \varepsilon @ l \rfloor \equiv \phi \quad \vdash \phi : (R, T)}{\vdash m[e_1]@e_2 \text{ as } \check{\phi} : \{E\} (m[w]@l : T), R \cdot \emptyset \{E\}} \\
& \text{APP} \\
& \frac{\text{sig}(f) = \Pi y_1, \dots, y_n. \{\check{E}_1\} \check{\Gamma}, \check{R} \cdot \check{\Delta} \{\check{E}_2\} \quad e_1 \Rightarrow v_1 \dots e_n \Rightarrow v_n \quad \rho = [v_1/y_1] \dots [v_n/y_n] \quad \rho(\{\check{E}_1\} \check{\Gamma}, \check{R} \cdot \check{\Delta} \{\check{E}_2\}) \Rightarrow \{E_1\} \Gamma, R \cdot \Delta \{E_2\} \quad E \models E_1}{\vdash f(e_1, \dots, e_n) : \{E\} \Gamma, R \cdot \Delta \{E \wedge E_2\}} \\
& \text{SEQ} \\
& \frac{\vdash \pi_1 : \{E_1\} \Gamma_1, R_1 \cdot \Delta_1 \{E_2\} \quad \vdash \pi_2 : \{E_2\} \Gamma_2, R_2 \cdot \Delta_2 \{E_3\}}{\vdash \pi_1; \pi_2 : \{E_1\} \Gamma_1; \Gamma_2, R_1; R_2 \cdot \Delta_1; \Delta_2 \{E_3\}}
\end{aligned}$$

SIG

$$\frac{C(f) = y_1, \dots, y_n, \mathbf{c} \quad \rho = [v_1/y_1] \cdots [v_n/y_n] \quad \rho(\{\check{E}_1\} \check{\Gamma}, \check{R} \cdot \check{\Delta} \{\check{E}_2\}) \Rightarrow \{E_1\} \Gamma, R \cdot \Delta \{E_2\} \quad \vdash \rho(\mathbf{c}) : \{E_1\} \Gamma, R \cdot \Delta \{E\} \quad E \models E_2}{f : \Pi y_1, \dots, y_n. \{\check{E}_1\} \check{\Gamma}, \check{R} \cdot \check{\Delta} \{\check{E}_2\}}$$

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 the judgement $\vdash \mathbf{c} : \{E_{\text{pre}}\} \Gamma, R \cdot \Delta \{E\}$ is derivable then $\mathbf{c} \Rightarrow \pi$ and:*

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

3.3 Confidentiality Examples

```
andtableycg(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;

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

p[o] := OT2(m[ag]@1, perm2(r[ag],(false,true)))@2

// p[o] : { c(r[ag]@2, {r[x]@2, r[y]@2, m[x]@1, m[y]@1}), r[ag]@2 }

out@1 := p[o]@1

// out@1 == s[x] and s[y]
encodegmw(in, i1, i2) {
  m[in]@i2 := (s[in] xor r[in])@i1;
  m[in]@i1 := r[in]@i1
}

andtablegmw(x, y, z) {
```

```

246     let r11 = r[z] xor (m[x] xor true) and (m[y] xor true) in
247     let r10 = r[z] xor (m[x] xor true) and (m[y] xor false) in
248     let r01 = r[z] xor (m[x] xor false) and (m[y] xor true) in
249     let r00 = r[z] xor (m[x] xor false) and (m[y] xor false) in
250     { row1 = r11; row2 = r10; row3 = r01; row4 = r00 }
251 }
252
253 andgmw(z, x, y) {
254     let table = andtablegmw(x,y,z) in
255     m[z]@2 := OT4(m[x],m[y],table,2,1);
256     m[z]@2 as ~((m[x]@1 xor m[x]@2) and (m[y]@1 xor m[y]@2)) xor r[z]@1;
257     m[z]@1 := r[z]@1
258 }
259
260 // and gate correctness postcondition
261 {} andgmw { m[z]@1 xor m[z]@2 == (m[x]@1 xor m[x]@2) and (m[y]@1 xor m[y]@2) }
262
263 // and gate type
264 andgmw :
265   Pi z,x,y .
266   {}
267   { { r[z]@1 },
268     (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 })} ),
269     m[z]@1 xor m[z]@2 == (m[x]@1 xor m[x]@2) and (m[y]@1 xor m[y]@2)}
270
271 xorgmw(z, x, y) {
272     m[z]@1 := (m[x] xor m[y])@1; m[z]@2 := (m[x] xor m[y])@2;
273 }
274
275 decodegmw(z) {
276     p["1"] := m[z]@1; p["2"] := m[z]@2;
277     out@1 := (p["1"] xor p["2"] )@1;
278     out@2 := (p["1"] xor p["2"] )@2
279 }
280
281 prot() {
282     encodegmw("x",2,1);
283     encodegmw("y",2,1);
284     encodegmw("z",1,2);
285     andgmw("g1", "x", "z");
286     xorgmw("g2", "g1", "y");
287     decodegmw("g2")
288 }
289
290 {} prot { out@1 == (s["x"]@1 and s["z"]@2) xor s["y"]@1 }

```

3.4 Integrity Examples

```

293 secopen(w1,w2,w3,i1,i2) {
294

```

```

295     pre(m[w1++"m"]@i2 == m[w1++"k"]@i1 + (m["delta"]@i1 * m[w1++"s"]@i2 /\
296         m[w1++"m"]@i2 == m[w1++"k"]@i1 + (m["delta"]@i1 * m[w1++"s"]@i2));
297     let locsum = macsum(macshare(w1), macshare(w2)) in
298     m[w3++"s"]@i1 := (locsum.share)@i2;
299     m[w3++"m"]@i1 := (locsum.mac)@i2;
300     auth(m[w3++"s"],m[w3++"m"],mack(w1) + mack(w2),i1);
301     m[w3]@i1 := (m[w3++"s"] + (locsum.share))@i1
302 }
303
304
305 _open(x,i1,i2){
306     m[x++"exts"]@i1 := m[x++"s"]@i2;
307     m[x++"extm"]@i1 := m[x++"m"]@i2;
308     assert(m[x++"extm"] == m[x++"k"] + (m["delta"] * m[x++"exts"]));
309     m[x]@i1 := (m[x++"exts"] + m[x++"s"]@i2
310 }`
311
312 _sum(z, x, y,i1,i2) {
313     pre(m[x++"m"]@i2 == m[x++"k"]@i1 + (m["delta"]@i1 * m[x++"s"]@i2 /\
314         m[y++"m"]@i2 == m[y++"k"]@i1 + (m["delta"]@i1 * m[y++"s"]@i2));
315     m[z++"s"]@i2 := (m[x++"s"] + m[y++"s"]@i2);
316     m[z++"m"]@i2 := (m[x++"m"] + m[y++"m"]@i2);
317     m[z++"k"]@i1 := (m[x++"k"] + m[y++"k"]@i1);
318     post(m[z++"m"]@i2 == m[z++"k"]@i1 + (m["delta"]@i1 * m[z++"s"]@i2)
319 }
320
321 sum(z,x,y) { _sum(z,x,y,1,2);_sum(z,x,y,2,1) }
322
323 open(x) { _open(x,1,2); _open(x,2,1) }
324
325
326 sum("a", "x", "d");
327 open("d");
328 sum("b", "y", "e");
329 open("e");
330 let xys =
331     macsum(macctimes(macshare("b"), m["d"]),
332         macsum(macctimes(macshare("a"), m["e"]),
333             macshare("c")))
334 let xyk = mack("b") * m["d"] + mack("a") * m["e"] + mack("c")
335
336 secopen("a", "x", "d", 1,2);
337 secopen("a", "x", "d", 2,1);
338 secopen("b", "y", "e", 1,2);
339 secopen("b", "y", "e", 2,1);
340 let xys =
341     macsum(macctimes(macshare("b"), m["d"]),
342         macsum(macctimes(macshare("a"), m["e"]),
343

```



```

344         macshare("c")))
345     in
346     let xyk = mack("b") * m["d"] + mack("d") * m["d"] + mack("c")
347     in
348     secreveal(xys,xyk,"1",1,2);
349     secreveal(maccsum(xys,m["d"] * m["e"]),
350         xyk - m["d"] * m["e"],
351         "2",2,1);
352     out@1 := (p[1] + p[2])@1;
353     out@2 := (p[1] + p[2])@2;
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392

```