1 Operational semantics

The rules that define the operational semantics of applied pi-calculus and ProVerif are adapted from [1]. The identifiers a, b, c, k and similar ones range over names, and x, y and z range over variables. As detailed in [1], set of symbols is also assumed for constructors and destructors such that f for a constructor and g for a destructor. Constructors are used to build terms. Therefore, the terms are variables, names, and constructor applications of the form $f(M_1, \ldots, M_n)$.

We use the constructors and destructors defined in [1] as an initial step to represent the cryptographic operations as depicted in Figure 1. We added other different constructors/destructors which are used to define our protocol. Constructors and destructors can be public or private. The public ones can be used by the adversary, which is the case when not stated otherwise. The private ones can be used only by honest participants.

Symmetric enc/dec:
Constructor: encryption of x with the shared secret key k, $senc(x, k)$
Destructor: decryption $sdec(senc(x, k), k) \rightarrow x$
Asymmetric enc/dec:
Constructor: encryption of x with the public key generation from a secret key k, $pk(k)$, $aenc(x, pk(k))$
Destructor: decryption $adec(aenc(x, pk(k)), k) \rightarrow x$
Signatures:
Constructors: signature of x with the secret key k, $sign(x, k)$
Destructors: signature verification using he public key generation from a secret key $k, pk(k), verify(sign(x, k), pk(k)) \rightarrow x$
One-way garbling function:
Constructors: garbling of x with the key $k, garble(x, k)$
Evaluation function:
Constructors: evaluation function of garbling of variables x, y, and z with the key k, $evaluate(garble(x, k), garble(y, k), garble(z, k))$
Commitment:
Constructors: committing x with a fresh nonce n k, $commit(x, n)$

Fig. 1. Constructors and destructors

The operational semantics used are presented in the Figure 2. A semantic configuration is a pair $\mathcal{E}$,$\mathcal{P}$ where the $\mathcal{E}$ is a finite set of names and $\mathcal{P}$ is a finite multiset of closed processes. The semantics of the calculus is defined by a reduction relation $\rightarrow$ on semantic configurations, shown in Figure 2. The process $event(M).\mathsf{P}$ executes the event $event(M)$ and then executes P. The input process $in(M, x).\mathsf{P}$ inputs a message, with x bound to tit, on channel M, and executes P. The output process $out(M, N).\mathsf{P}$ outputs the message N on the channel M and then executes P.

The nil process 0 does nothing. The process $\mathsf{P}|\mathsf{Q}$ is the parallel composition of P and Q. The replication $!\mathsf{P}$ represents an unbounded number of copies of P

$(\mathit{Nill})\quad \mathcal{E}, \mathcal{P} \cup \{0\} \rightarrow \mathcal{E}, \mathcal{P}$

$(\mathit{Repl})\quad \mathcal{E}, \mathcal{P} \cup \{!\mathsf{P}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, !\mathsf{P}\}$

$(\mathit{Par})\quad \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}|\mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, \mathsf{Q}\}$

$(\mathit{Par})\quad \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}|\mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, \mathsf{Q}\}$

$(\mathit{New})\quad \mathcal{E}, \mathcal{P} \cup \{(\mathit{newa})\mathsf{P}\} \rightarrow \mathcal{E} \cup \{a'\}, \mathcal{P} \cup \{\mathsf{P}\{a'/a\}\}$ where $a' \notin \mathcal{E}$

$(I/O)\quad \mathcal{E}, \mathcal{P} \cup \{\mathit{out}(c, M).\mathsf{Q}, \mathit{in}(c, x).\mathsf{P}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{Q}, \mathsf{P}\{M/x\}\}$

$(\mathit{Cond}1)\quad \mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } \mathsf{P} \text{ else } \mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}\}$ if $M = N$

$(\mathit{Cond}2)\quad \mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } \mathsf{P} \text{ else } \mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{Q}\}$ if $M \neq N$

$(\mathit{Let})\quad \mathcal{E}, \mathcal{P} \cup \{\text{let } x = g(M_1, \ldots, M_n) \text{ in } \mathsf{P} \text{ else } \mathsf{Q}\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}\{M'/x\}\}$
if $g(M_1, \ldots, M_n) \rightarrow M'$

Fig. 2. Operational semantics

in parallel. $(\mathit{newa})\mathsf{P}$ creates a new name a and then executes P. The conditional if $M = N$ then P else Q executes P if M and N reduce to the same term at runtime; otherwise, it executes Q. Finally, let $x = M$ in P as syntactic sugar for $\mathsf{P}\{M/x\}$ which is the process obtained from P by replacing every occurrence of x with M. As usual, we may omit an else clause when it consists of 0.

2 Protocol model

In this section we model the *Qese* protocol presented in the submitted paper and depicted in Figure 3 (detailed in Garbled-integrity proverif script; in the formal specification folder on GitHub).

Provider(CSP)		**Broker**(B)
$knowledge:$		$knowledge:$
$\mathsf{pk_B}, \mathsf{k}^{auth}_{\mathsf{CSP}}, t^1_{\mathsf{CSP}}, \ldots, t^n_{\mathsf{CSP}}$		$\mathsf{sk_B}, \mathsf{pk_B}, \mathsf{k}^{auth}_{\mathsf{CSP}}, w^1_{\mathsf{C}}, \ldots, w^n_{\mathsf{C}}$
		$N_{\mathsf{B}} \in \{0,1\}^{\beta}$
	$\xleftarrow[Message_1]{G^{w^1_{\mathsf{C}}}}$	$G^{w^1_{\mathsf{C}}} := commit(w^1_{\mathsf{C}}, N_{\mathsf{B}})$
$\mathsf{k_{Gar}} \leftarrow_{\$} \mathsf{KGen}(1^n)$		
$\mathsf{GarEnc_B} \leftarrow_{\$} garble(G^{w^1_{\mathsf{C}}}, \mathsf{k_{Gar}})$		
$\mathsf{GarEnc_k} \leftarrow_{\$} garble(\mathsf{Enc_k}, \mathsf{k_{Gar}})$		
$\mathsf{GarEnc}_{\mathsf{CSP}_{t^1_{\mathsf{CSP}}}} \leftarrow_{\$} garble(t^1_{\mathsf{CSP}}, \mathsf{k_{Gar}})$		
$E^{\mathsf{CSP}}_{garble} := evaluate(\mathsf{GarEnc_B}, \mathsf{GarEnc_k}, \mathsf{GarEnc}_{\mathsf{CSP}_{t^1_{\mathsf{CSP}}}})$		
$c^{enc}_{\mathsf{CSP}} \leftarrow_{\$} \mathsf{Enc}(\mathsf{pk_B}, \mathsf{GarEnc_B}, \mathsf{GarEnc_k}, \mathsf{GarEnc}_{\mathsf{CSP}_{t^1_{\mathsf{CSP}}}}, E^{\mathsf{CSP}}_{garble}, G^{w^1_{\mathsf{C}}})$	$\xrightarrow[Message_2]{c^{enc}_{\mathsf{CSP}}}$	
		$\mathsf{GarEnc_B}, \mathsf{GarEnc_k}, \mathsf{GarEnc}_{\mathsf{CSP}_{t^1_{\mathsf{CSP}}}}, E^{\mathsf{CSP}}_{garble}, G^{w^1_{\mathsf{C}}} \leftarrow_{\$} \mathsf{Dec}(\mathsf{sk_B}, c^{enc}_{\mathsf{CSP}})$
		$G^{w^1_{\mathsf{C}}} \overset{?}{=} commit(w^1_{\mathsf{C}}, N_{\mathsf{B}})$
		Abort if not equal
		$E^{\mathsf{B}}_{garble} := evaluate(\mathsf{GarEnc_B}, \mathsf{GarEnc_k}, \mathsf{GarEnc}_{\mathsf{CSP}_{t^1_{\mathsf{CSP}}}})$
		$E^{\mathsf{B}}_{garble} \overset{?}{=} E^{\mathsf{CSP}}_{garble}$
	$\xleftarrow[Message_3]{E^{\mathsf{B}}_{garble}}$	Abort if not equal
$E^{\mathsf{B}}_{garble} \overset{?}{=} E^{\mathsf{CSP}}_{garble}$		
Abort if not equal		

Fig. 3. Tokens encryption and secure computation protocol (QeSe)

$\mathsf{P_B}(sk_\mathsf{B}, pk_\mathsf{B}, m_\mathsf{B}) = !in(c, m).(new b) event(e_1(m_\mathsf{B}, b, commit(m_\mathsf{B}, b))).$
$out(c, commit(m_\mathsf{B}, b)).in(c, m'').let((x_\mathsf{B}, x_f, x_\mathsf{CSP}, m_x) = adec(m'', sk_\mathsf{B}))in$
$if m_x = commit(m_\mathsf{B}, b) then$
$event(e_\mathsf{B}(x_\mathsf{B}, x_f, x_\mathsf{CSP}, evaluate(x_\mathsf{B}, x_f, x_\mathsf{CSP}))).out(c, evaluate(x_\mathsf{B}, x_f, x_\mathsf{CSP}))$

$\mathsf{P_{CSP}}(pk_\mathsf{B}, m_f, m_\mathsf{CSP}) = in(c, m').let((y_\mathsf{B} = garble(m', k))|$
$(y_f = garble(m_f, k))|(y_\mathsf{CSP} = garble(m_\mathsf{CSP}, k)))in$
$event(e_2(y_\mathsf{B}, y_f, y_\mathsf{CSP}, sk_\mathsf{CSP})).out(c, (senc((y_\mathsf{B}, y_f, y_\mathsf{CSP}, m'), sk_\mathsf{CSP})).in(c, m''')$
$if m''' = evaluate(y_\mathsf{B}, y_f, y_\mathsf{CSP}) then$
$event(e_\mathsf{CSP}(y_\mathsf{B}, y_f, y_\mathsf{CSP}, m'''))$

$\mathsf{P}(new m_f)(new m_\mathsf{CSP})(new m_\mathsf{B})(new sk_\mathsf{B}) let pk_\mathsf{B} = pk_{sk_\mathsf{B}} in$
$out(c, pk_\mathsf{B}).\mathsf{P_B}(sk_\mathsf{B}, pk_\mathsf{B}, m_\mathsf{B})|\mathsf{P_{CSP}}(pk_\mathsf{B}, m_f, m_\mathsf{CSP})$

The channel c is public so that the adversary can send, replay and get any messages sent over it. We use a single public channel and not two or more channels because the adversary could take a message from one channel and relay it on another channel, thus removing any difference between the channels. The process P begins with the creation of the secret and public keys of B, and the creation of messages $m_f, m_\mathsf{CSP}, m_\mathsf{B}$ The public key is output on channel c to model that the adversary has it in its initial knowledge. Then the protocol itself starts: $\mathsf{P_B}$ represents the B, $\mathsf{P_{CSP}}$ represents CSP. Both principals can run an unbounded number of sessions, so $\mathsf{P_B}$ and $\mathsf{P_{CSP}}$ start with replications.

We consider that B first inputs a message containing the encrypted tokens along with the authentication secret. Once B validates the authentication secret, it stores the encrypted tokens and starts the protocol run by choosing a nonce b, and executing the event $e_1(m_\mathsf{B}, b, commit(m_\mathsf{B}, b))$, where m_B is initially added to the B knowledge. Intuitively, this event records that B sent $Message_1$ of the protocol. Event e_1 is placed before the actual output of $Message_1$; this is necessary for the desired correspondences to hold: if event e_1 followed the output of $Message_1$, we would not be able to prove that event e_1 must have been executed, even though $Message_1$ must have been sent, because $Message_1$ could be sent without executing event e_1, as stated in [1]. The situation is similar for events e_2, e_3 and e_4.

Next, B receives the garbling of CSP's inputs as well as the garbling of the committed messages encrypted with its public key. B decrypts the message using its secret key sk_B. If decryption succeeds B checks if the message has the right form using the pattern-matching construct $let((x_\mathsf{B},\ x_f, x_\mathsf{CSP}, = m_\mathsf{B}) = adec(m'', sk_\mathsf{B}))in.$ Then B executes the event $e_\mathsf{B}(x_\mathsf{B}, x_f, x_\mathsf{CSP})$, to record that it has received $Message_2$ and sent $Message_3$ of the protocol. Finally, B sends the last message of the protocol $evaluate(x_\mathsf{B}, x_f, x_\mathsf{CSP})$.

After sending this message, B executes some actions needed only for specifying properties of the protocol. When the received message $m_x = \mathit{commit}(m_{\mathsf{B}}, b)$, that is, when the session is between B and CSP, B executes the event $e_3(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}})$, to record that B ended a session of the protocol, with the participant (CSP), which is authenticated using the authentication key. B also outputs the evaluation function output $\mathit{evaluate}(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}})$.

The process $\mathsf{P_{CSP}}$ proceeds similarly: it executes the protocol, with the additional event $e_2(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}}, sk_{\mathsf{CSP}})$ to record that $Message_1$ has been received and $Message_2$ has been sent by CSP, in a session with the participant of public key pk_{B} and the received message m'. After finishing the protocol itself, when $m''' = \mathit{evaluate}(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}})$, that is, when the session is between B and CSP, $\mathsf{P_{CSP}}$ executes the event $e_{\mathsf{CSP}}(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}}, m''')$, to record that CSP finished the protocol, and outputs m'''.

The events will be used in order to formalize integrity. For example, we formalize that, if CSP ends a session of the protocol $e_{\mathsf{CSP}}(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}}, m''')$, then (a) B has started a session of the protocol by committing m_{B} with the nonce n_{B}, and (b) CSP outputs the evaluation function $\mathit{evaluate}(y_{\mathsf{B}}, y_f, y_{\mathsf{CSP}})$. Furthermore, B ends a session of the protocol, then (a) CSP has already garbled the B's committed input message, and (b) B outputs the evaluation function $\mathit{evaluate}(x_{\mathsf{B}}, x_f, x_{\mathsf{CSP}})$.

References

1. B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.