MrCrypt: Static Analysis for Secure Cloud Computations

UCLA Technical Report #130012

Mohsen Lesani Rupak Majumdar Todd Millstein Sai Deep Tetali
Computer Science Department
University of California, Los Angeles

Contents

1 Encryption Calculus

This section formally defines the encryption scheme inference problem on an extended simply-typed lambda calculus, formalizes our solution, and proves various correctness and security properties of the approach.

Our formalism is parameterized by a set $\mathcal{M}$ of arithmetic operations and a set $\mathcal{R}$ of logical predicates, whose union we denote $\mathcal{O}$. The formalism is also parameterized by a lattice L of encryption schemes, each of which supports some subset of the operations in $\mathcal{O}$, with associated partial order $\sqsubseteq$. We assume that if $l_1 \sqsubseteq l_2$ and encryption scheme l_2 supports some operation $f \in \mathcal{O}$, then l_1 also supports that operation. Also, for each operation $f \in \mathcal{O}$ we assume there is a unique maximal element of L that supports f, which we denote l_f.

In our implementation, $\mathcal{M} = \{+, \times\}$ and $\mathcal{R} = \{<, =, >\}$ and We have the set of encryption schemes $L = \{\mathit{RAND}, \mathit{OP}, \mathit{DET}, \mathit{MH}, \mathit{AH}, \mathit{FHE}\}$. *RAND* encryption supports no operations, *OP* supports $\{=, <, >\}$, *DET* supports $\{=\}$, *MH* supports $\{\times\}$, *AH* supports $\{+\}$, *FHE* supports all of these operations. We define the encryption lattice on L as follows: $\mathit{FHE} \sqsubseteq \mathit{OP}$, $\mathit{FHE} \sqsubseteq \mathit{MH}$, $\mathit{FHE} \sqsubseteq \mathit{AH}$, $\mathit{OP} \sqsubseteq \mathit{DET}$, $\mathit{DET} \sqsubseteq \mathit{RAND}$, $\mathit{MH} \sqsubseteq \mathit{RAND}$, $\mathit{AH} \sqsubseteq \mathit{RAND}$. In this lattice, *FHE* is $\bot$ and *RAND* is $\top$.

1.1 Source Programs

We define the set of expressions as follows

$$\begin{array}{rcll} e & ::= & e_1\ e_2 \ \mid\ e_1\ f\ e_2 \ \mid & \text{Program} \\ & & v \ \mid\ x & \\ v & ::= & \lambda x{:}\rho.e \ \mid\ n \ \mid\ n_l & \text{Value} \end{array}$$

In our implementation, $\mathcal{M} = \{+, \times\}$ and $\mathcal{R} = \{<, =, >\}$. The set of types is defined as follows:

$$\begin{array}{rcll} \tau & ::= & \mathit{Int} \ \mid\ \alpha \ \mid\ \rho \rightarrow \rho & \text{Type} \\ \kappa & ::= & l \ \mid\ \circ \ \mid\ \gamma & \text{Qualifier} \\ \rho & ::= & \kappa\ \tau & \text{Qualified Type} \end{array}$$

A program is a closed function. Given the program, the type inferencer will automatically infer the optimum encryption schemes for the input variables. Note that the type of a lambda abstraction parameter can be a type variable and the type inferencer can infer the type. Constants can appear in programs. n is a natural number. n_l denotes an encrypted value of the natural number n with the encryption scheme l. Instead of encoding constants in the program, they can be given as inputs to the program so that the type inferencer automatically infers their encryption schemes.

α denotes a type variable, γ denotes a qualifier variable. $\circ$ is the qualifier for unencrypted data. As any operation is supported on unencrypted data, $\sqsubseteq$ is extended such that $\forall f : \circ \sqsubseteq l_f$

We define $decr(e)$ that decrypts the the encrypted numbers in e, as follows: $decr(e_1\, e_2) = decr(e_1)\, decr(e_2)$, $decr(e_1\, f\, e_2) = decr(e_1)\, f\, decr(e_2)$, $decr(\lambda x{:}\rho.e) = \lambda x{:}\rho.decr(e)$, $decr(n_l) = n$, $decr(n) = n$ and $decr(x) = x$.

1.2 Plaintext Domain

1.2.1 Operational Semantics

$$\begin{array}{rcll} R & ::= & [\,]\ \mid & \text{Reduction Context} \\ & & R\ e\ \mid\ v\ R\ \mid & \\ & & R\ f\ e_2\ \mid\ v\ f\ R & \end{array}$$

We characterize the transitions of these operations as follows:

App-Trans
$$R[(\lambda x{:}\rho.e)\ v] \rightarrow R[e[x \mapsto v]]$$

Rel-Trans
$$\frac{n'' = \begin{cases} 1 & \text{if } (n\ f\ n') \\ 0 & \text{if } \neg(n\ f\ n') \end{cases} \qquad f \in \mathcal{R}}{R[n\ f\ n'] \rightarrow_p R[n'']}$$

Math-Trans
$$\frac{n\ f\ n' = n'' \qquad f \in \mathcal{M}}{R[n'\ f\ n''] \rightarrow_p R[n'']}$$

1.3 Encrypted Domain

1.3.1 Typing

The typing environment is a mapping from a set of variables x to types ρ. The typing rules are defined as follows: The judgments are of the form $\Gamma \vdash e\colon \rho$ i.e. in the type environment Γ, the expression e has type ρ.

Q-Lam
$$\frac{\Gamma[x \mapsto \rho_1] \vdash e \colon \rho_2}{\Gamma \vdash \lambda x{:}\rho_1.e \colon\ \circ\,(\rho_1 \to \rho_2)}$$

Q-App
$$\frac{\Gamma \vdash e_1 \colon \kappa\ (\rho_1 \to \rho_2) \qquad \Gamma \vdash e_2 \colon \rho_1}{\Gamma \vdash e_1\ e_2 \colon \rho_2}$$

Q-Math
$$\frac{\Gamma \vdash e' \colon \kappa\ \mathit{Int} \qquad \Gamma \vdash e'' \colon \kappa\ \mathit{Int} \qquad \kappa \sqsubseteq l_f \qquad f \in \mathcal{M}}{\Gamma \vdash e'\ f\ e'' \colon \kappa\ \mathit{Int}}$$

Q-Rel
$$\frac{\Gamma \vdash e' \colon \kappa\ \mathit{Int} \qquad \Gamma \vdash e'' \colon \kappa\ \mathit{Int} \qquad \kappa \sqsubseteq l_f \qquad f \in \mathcal{R}}{\Gamma \vdash e'\ f\ e'' \colon\ \circ\ \mathit{Int}}$$

Q-Int-l
$$\Gamma \vdash n_l \colon l\ \mathit{Int}$$

Q-Int
$$\Gamma \vdash n \colon\ \circ\,\mathit{Int}$$

Q-Var
$$\frac{\Gamma(x) = \rho}{\Gamma \vdash x \colon \rho}$$

The typing rules enforce that operations should be applied to the operands of the same encryption scheme and that the encryption scheme should support the operation. Also for assertion expressions, the encryption of the asserted expression is enforced to be at least as strong as the the asserted scheme.

1.3.2 Operational Semantics

$$\begin{array}{lll} R & ::= & [\,]\ \mid \\ & & R\ e\ \mid\ v\ R\ \mid \\ & & R\ f\ e_2\ \mid\ v\ f\ R \end{array} \qquad \text{Reduction Context}$$

We characterize the transitions of these operations as follows:
The transition rules are defined as follows.

Q-App-Trans
$$R[(\lambda x{:}\rho.e)\ v] \to R[e[x \mapsto v]]$$

Q-Math-Trans
$$\frac{n\ f\ n' = n'' \qquad f \in \mathcal{M} \qquad l \sqsubseteq l_f}{R[n_l\ f\ n'_l] \to R[n''_l]}$$

Q-Rel-Trans
$$\frac{n'' = \begin{cases} 1 & \text{if } (n\ f\ n') \\ 0 & \text{if } \neg(n\ f\ n') \end{cases} \qquad f \in \mathcal{R} \qquad l \sqsubseteq l_f}{R[n_l\ f\ n'_l] \to R[n'']}$$

1.3.3 Type Inference

The judgments are of the form $\Gamma \vdash e\colon \rho; C; \mathcal{X}$ i.e. in the type environment Γ, the expression e has type ρ under the constraints C. The set of variables $\mathcal{X}$ keeps track of the variables that are introduced in each subderivarion. The set of constraints is defined as

$$\begin{array}{rcll} C & ::= & \{\tau_1 = \tau_2\} \;\mid & \text{Constraint} \\ & & \{\kappa_1 = \kappa_2\} \;\mid\; \{\gamma \sqsubseteq l\} \;\mid & \\ & & C_1 \cup C_2 & \end{array}$$

Equality constraints of the form $\kappa_1\ \tau_1 = \kappa_2\ \tau_2$ can be desugared to $\{\kappa_1 = \kappa_2, \tau_1 = \tau_2\}$.

The type inference rules are defined as:

Q-Lam-Inf

$$\frac{\begin{array}{c}\Gamma[x \mapsto \rho_1] \vdash e\colon \rho_2; C; \mathcal{X} \\ C' = C \cup \{\gamma = \circ, \alpha = \rho_1 \to \rho_2\} \\ \mathcal{X}' = \mathcal{X} \cup \{\gamma, \alpha\} \qquad \gamma, \alpha \text{ fresh}\end{array}}{\Gamma \vdash \lambda x{:}\rho_1.e\colon (\gamma\ \alpha); C'; \mathcal{X}'}$$

Q-App-Inf

$$\frac{\begin{array}{c}\Gamma \vdash e_1\colon \rho_1; C_1; \mathcal{X}_1 \qquad \Gamma \vdash e_2\colon \rho_2; C_2; \mathcal{X}_2 \\ C = C_1 \cup C_2 \cup \{\rho_1 = \gamma\ (\rho_2 \to \gamma'\ \alpha)\} \\ \mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \{\gamma, \gamma', \alpha\} \\ \mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap FV(\rho_2) = \mathcal{X}_2 \cap FV(\rho_1) = \emptyset \\ \gamma, \gamma', \alpha \text{ fresh}\end{array}}{\Gamma \vdash e_1\ e_2\colon (\gamma'\ \alpha); C; \mathcal{X}}$$

Q-Math-Inf

$$\frac{\begin{array}{c} f \in \mathcal{M} \\ \Gamma \vdash e_1\colon (\kappa_1\ \tau_1); C_1; \mathcal{X}_1 \qquad \Gamma \vdash e_2\colon (\kappa_2\ \tau_2); C_2; \mathcal{X}_2 \\ C = C_1 \cup C_2 \cup \{\kappa_1 = \kappa_2 = \gamma, \gamma \sqsubseteq l_f, \tau_1 = \tau_2 = Int = \alpha\} \\ \mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \{\gamma, \alpha\} \\ \mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap FV(\kappa_2\ \tau_2) = \mathcal{X}_2 \cap FV(\kappa_1\ \tau_1) = \emptyset \\ \gamma, \alpha \text{ fresh}\end{array}}{\Gamma \vdash (e_1\ f\ e_2)\colon (\gamma\ \alpha); C; \mathcal{X}}$$

Q-Rel-Inf

$$\frac{\begin{array}{c} f \in \mathcal{R} \\ \Gamma \vdash e_1 \colon (\kappa_1\ \tau_1); C_1; \mathcal{X}_1 \qquad \Gamma \vdash e_2 \colon (\kappa_2\ \tau_2); C_2; \mathcal{X}_2 \\ C = C_1 \cup C_2 \cup \{\kappa_1 = \kappa_2, \gamma = \circ, \kappa_1 \sqsubseteq l_f, \tau_1 = \tau_2 = Int = \alpha\} \\ \mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \{\gamma, \alpha\} \\ \mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap FV(\kappa_2\ \tau_2) = \mathcal{X}_2 \cap FV(\kappa_1\ \tau_1) = \emptyset \\ \gamma, \alpha \text{ fresh} \end{array}}{\Gamma \vdash (e_1\ f\ e_2) \colon (\gamma\ \alpha); C; \mathcal{X}}$$

Q-Var-Inf

$$\frac{\Gamma(x) = \rho}{\Gamma \vdash x \colon \rho; \emptyset; \emptyset}$$

Q-Int-l-Inf

$$\frac{\begin{array}{c} C = \{\gamma = l, \alpha = Int\} \qquad \mathcal{X} = \{\gamma, \alpha\} \\ \gamma, \alpha \text{ fresh} \end{array}}{\Gamma \vdash n_l \colon (\gamma\ \alpha); C; \mathcal{X}}$$

Q-Int-Inf

$$\frac{\begin{array}{c} C = \{\gamma = \circ, \alpha = Int\} \qquad \mathcal{X} = \{\gamma, \alpha\} \\ \gamma, \alpha \text{ fresh} \end{array}}{\Gamma \vdash n \colon (\gamma\ \alpha); C; \mathcal{X}}$$

Encryption Scheme Inference. The encryption scheme inference problem is defined as the following type inference problem. Consider a program (a closed expression) e. Let ρ, C and $\mathcal{X}$ be a type, a set of constraints and a set of variables such that $\emptyset \vdash e \colon \rho; C; \mathcal{X}$. Let the mapping σ be a solution for C. An encryption scheme for e is the restriction of σ to the qualifier variables of e.

1.4 Soundness

Lemma 1 (Progress). *If $\emptyset \vdash e \colon \rho$, then either e is a value or there is some e' such that $e \to e'$.*

Lemma 2 (Preservation). *If $\Gamma \vdash e \colon \rho$ and $e \to e'$ then $\Gamma \vdash e' \colon \rho$.*

Intuitively, these two lemma state that a well-typed program never *gets stuck* during transition. This means that, at runtime, operations are applied to operands of the same encryption scheme.

Now, we show that the computation in the encrypted domain parallels the computation in the plaintext domain.

Lemma 3 (Encryption Domain Soundness). *If $e \to e'$, then $decr(e) \to_p decr(e')$.*

Intuitively this means that any transition in the encrypted domain has a corresponding transition in the plaintext domain.

Lemma 4 (Encryption Domain Completeness). *If $\emptyset \vdash e_1 \colon \rho$, $e'_1 = decr(e_1)$ and $e'_1 \to_p e'_2$, then $e_1 \to e_2$ and $e'_2 = decr(e_2)$.*

Intuitively, this means that any transition in plaintext domain has a corresponding transition in encrypted domain.

Lemma 5 (Soundness of Type Inference). *If $\Gamma \vdash e \colon \rho; C; \mathcal{X}$ and σ is a substitution such that $\sigma(C)$ is valid, then $\sigma(\Gamma) \vdash \sigma(e) \colon \sigma(\rho)$.*

The soundness of type inference intuitively means that once the inferred types are applied to the program, the program is well-typed.

A write $\sigma \setminus \mathcal{X}$ for a substitution that is undefined for all the variables in $\mathcal{X}$ and otherwise behaves like σ.

Lemma 6 (Completeness of Type Inference). *If $\Gamma \vdash e \colon \rho; C; \mathcal{X}$ and there is a substitution σ such that $\sigma(\Gamma) \vdash \sigma(e) \colon \rho'$ and $dom(\sigma) \cap \mathcal{X} = \emptyset$ then there is a substitution σ' such that $\sigma'(C)$ is valid and $\sigma'(\rho) = \rho'$ (and $\sigma' \setminus \mathcal{X} = \sigma$).*

The completeness of type inference intuitively means that if there is a typing for a program, then type inference can find it.

1.5 Security Guarantees

We assume an honest-but-curious adversary model, where the server observes the data, the program, and the program execution and can perform polynomial-time computation over the observations. However, the server does not change the data or the computation. One caveat is that the server should run in polynomial time in the size of the data and the input, but not in the potentially exponential program trace. If we allow the adversary to run in time polynomial in the program trace, it may be able to execute an exponentially long computation in the security parameter, and so to decrypt all the encrypted values trivially.

We formalize our security guarantees in terms of *indistinguishability* [5]. Indistinguishability is formalized using an *adversary* $A = (A_1, A_2)$, performing a sequence of two (potentially randomized) polynomial-time algorithms. Initially keys $(pk, sk) = K(\lambda)$ are generated based on a security parameter λ. First, algorithm A_1 takes as input the public key pk and outputs two plaintext messages x_0 and x_1, together with some additional state information s. Next, a bit $b \in \{0, 1\}$ is chosen at random, and message x_b is encrypted as a challenge ciphertext y using pk. Finally, algorithm A_2 runs on (y, s) and has to guess the bit b. The advantage of the adversary is defined as

$$Adv_{\mathcal{E}}(A) = \Pr[A_2(y, s) = b] - \frac{1}{2}$$

where the random variables are distributed uniformly.

An encryption scheme $\mathcal{E} = (K, E, D)$ satisfies single-use *indistinguishability* against chosen plaintext attacks (IND-CPA) if for each adversary A we have that $Adv_{\mathcal{E}}(A)$ is negligible (recall that a function $f(n)$ is negligible if $|f(n)| < \frac{1}{poly(n)}$ for all sufficiently large n). Intuitively, a polynomial-time adversary cannot identify the plaintext from a ciphertext with advantage significantly better than that obtained by flipping a coin. For example, it is known that the El Gamal and Paillier cryptosystems satisfy IND-CPA.

Unfortunately, IND-CPA is too strong a requirement for deterministic encryption schemes: for example, the adversary can store the encryptions of x_0 and x_1 and compare the challenge ciphertext y against the stored ciphertexts. Similarly, IND-CPA is too strong for order-preserving schemes. Thus, one defines weaker notions of indistinguishability for such schemes. We omit detailed definitions (see, e.g., [2, 3, 4]), but assume that each individual encryption scheme $\mathcal{E}$ has an associated indistinguishability property IND($\mathcal{E}$).

In our context, we have a set of inputs $x_1, \ldots, x_n$ to the program, and use possibly different encryption schemes $\mathcal{E}_1, \ldots, \mathcal{E}_n$ for them. We ask, given that each scheme $\mathcal{E}_i$ satisfies IND($\mathcal{E}_i$), what we can guarantee about the full encrypted data. To do this, we define the notion of *program-indistinguishability* for a tuple of encryption schemes. Intuitively, the adversary now chooses two sequences of plaintexts, according to possible restrictions placed by the IND conditions. Now one of the two is chosen at random and componentwise encoded using its encryption scheme. The adversary has to guess which of the two sequences was encoded by looking at the encrypted vector. Notice that we do not consider the encrypted program in the definition, since the adversary can perform an arbitrary polynomial-time computation, in particular, it can run the program for a polynomial number of steps. The following theorem generalizes a result from [1].

Lemma 7. *Given encryption schemes $\mathcal{E}_i$ satisfying IND($\mathcal{E}_i$) for $i = 1, \ldots, n$, $(\mathcal{E}_1, \ldots, \mathcal{E}_n)$ is program-indistinguishable.*

Thus, MrCrypt provides a security guarantee that is as strong as the individual encryption schemes used for each data item.

2 Proofs

2.1 Helper Lemmas

Lemma 8 (Inversion of Typing).

- *If* $\Gamma \vdash \lambda x{:}\rho_1.e\ : \rho$ *then* $\rho = \circ\ (\rho_1 \to \rho_2)$ *for some* ρ_2 *with* $\Gamma, x{:}\rho_1 \vdash e{:}\ \rho_2$.
- *If* $\Gamma \vdash n_l\ : \rho$ *then* $\rho = l\ int$.
- *If* $\Gamma \vdash n\ : \rho$ *then* $\rho = int$.

Proof. Immediate form typing derivation rules. □

Lemma 9 (Canonical Forms).

- *If* v *is a value of type* $\rho_1 \to \rho_2$ *then* $v = \lambda x{:}\rho_1.e$ *for some* e.
- *If* v *is a value of type* $l\ int$ *then* $v = n_l$ *for some* n.
- *If* v *is a value of type* int *then* $v = n$ *for some* n.

Proof. Immediate by case analysis on the structure of v and using the inversion lemma, Lemma 8. □

Lemma 10 (Type Preservation Under Substitution). *If* $\Gamma, x{:}\rho \vdash e{:}\ \rho'$ *and* $\Gamma \vdash v{:}\ \rho$, *then* $\Gamma \vdash e[x \mapsto v]{:}\ \rho'$.

Proof. Immediate by induction on the derivation of $\Gamma, x{:}\rho \vdash e{:}\ \rho'$. □

Lemma 11 (Inversion of Decryption).

- $decr(e) = \lambda x{:}\rho_1.e'_1$ *then* $e = \lambda x{:}\rho_1.e_1$ *where* $decr(e_1) = e'_1$.
- $decr(e) = n$ *then either* $e = n$ *or* $e = n_l$ *for some* l.

Proof. Immediate form the definition of *decr*. □

2.2 Main Lemmas

Lemma 1 (Progress)
Hypothesis:
    $\emptyset \vdash e\colon \rho$
Conclusion:
    either e is a value
    or there is some e' such that $e \to e'$

Proof.
Induction on the derivation of $\emptyset \vdash e\colon \rho$.
Case the rule Q-Lam:
    $\lambda x{:}.e$ is a value.
Case the rule Q-App:
    e_1 and e_2 are typed.
    Induction hypothesis is applied to e_1 and e_2.
    If both are values, using the canonical lemma (Lemma 9),
    the rule Q-App-Trans can be applied.
    Otherwise, by reduction contexts $R\ e$ and $v\ R$,
    the transitions of e_1 or e_2, yield transitions for $e_1\ e_2$.
Case the rule Q-Math:
    This is similar to the rule Q-App case, except that
    for the case where both e and e' are values,
    the required condition $l \sqsubseteq l_f$ of the rule Q-Math-Trans
    is given by the rule Q-Math.
Case the rule Q-Rel:
    Similar to the case of the rule Q-Math.
Case the rule Q-Int-L:
    n_l is a value.
Case the rule Q-Int:
    n is a value.
Case the rule Q-Var:
    A variable in not typed under empty environment.

□

Lemma 2 (Preservation)
Hypothesis:
    $\Gamma \vdash e\colon \rho$
    $e \to e'$
Conclusion:
    $\Gamma \vdash e'\colon \rho$

Proof.
Straightforward induction on the derivation of $\Gamma \vdash e\colon \rho$ and then case analysis on the final rule in the derivation of $e \to e'$.
Case the rule Q-Lam:
    No reduction rule can be applied to $\lambda x{:}.e$.
Case the rule Q-App:
    e_1 and e_2 are typed.
    If the transition is by the rule Q-App-Trans,
        As the type of e_1 is a function type, it is
        typed by the rule Q-Lam. From the premise of
        the rule Q-Lam and type preservation under substitution
        lemma, Lemma 10, we get the result.
    Otherwise, the transitions are in reduction contexts $R\ e$ or $v\ R$,
        By the induction hypothesis, the type of e_1 or e_2 is preserved
        after the transition. Thus, the rule Q-App yields the result.
Case the rule Q-Op-Math:
    This is similar to the case of the rule Q-App, except that
    for the case where both e and e' are values,
    the rule Q-Int-L yields the result.
Case the rule Q-Op-Rel:
    Similar to the case of the rule Q-Op-Math.
Case the rule Q-Int-L:
    No reduction rule can be applied to n_l.
Case the rule Q-Int:
    No reduction rule can be applied to n.
Case the rule Q-Var:
    No reduction rule can be applied to a variable x.

□

Lemma 3 (Encryption Domain Soundness)
Hypothesis:
    $e \to e'$
Conclusion:
    $decr(e) \to_p decr(e')$

Proof.
Straightforward induction on the derivation of $e \to e'$.
Case the rule Q-App-Trans
    Immediate from the rule App-Trans.
Case the rule Q-Math-Trans

Immediate from the rule Math-Trans.

Case the rule Q-Rel-Trans

Immediate from the rule Rel-Trans.

□

Lemma 4 (Encryption Domain Completeness)

Hypothesis:

$\emptyset \vdash e_1 : \rho$,

$e'_1 = decr(e_1)$

$e'_1 \rightarrow_p e'_2$

Conclusion:

$e_1 \rightarrow e_2$

$e'_2 = decr(e_2)$

Proof.

Straightforward induction on the derivation of $\emptyset \vdash e_1 : \rho$.

Case the rule Q-Lam:

$decr(e_1)$ is a lambda term. It cannot be reduced.

Case the rule Q-App:

$e_1 = e_{11}\ e_{12}$

$e'_1 = decr(e_1) = e'_{11}\ e'_{12}$ where

$e'_{11} = decr(e_{11})$ and $e'_{12} = decr(e_{12})$

If $e'_1 \rightarrow e'_2$ is by the rule App-Trans,

We have that $e'_{11} = \lambda x{:}\rho.e'$, $e'_{12} = v'$, $e'_2 = e'[x \mapsto v']$

By the inversion lemma, Lemma 11, we have

$e_{11} = \lambda x{:}\rho.e$ such that $decr(e) = e'$ and

$e_{12} = v$ such that $decr(v) = v'$.

Thus, $e'_2 = e'[x \mapsto v'] = decr(e)[x \mapsto decr(v)]$

By the rule Q-App-Trans, $e_1 \rightarrow e_2$ where $e_2 = e[x \mapsto v]$

which yields the result.

Otherwise,

Either $e'_{11} = decr(e_{11})$ or $e'_{12} = decr(e_{12})$ makes a transition.

The result follow from the induction hypothesis.

Case the rule Q-Math:

Similar to the case of the rule Q-App.

The important difference is that

in the case that $e'_1 \rightarrow e'_2$ is by the rule Math-Trans, the fact that

the two operands have the same encryption scheme l

comes from the premises of the rule Q-Math.

Case the rule Q-Rel:

Similar to the case of the rule Q-Math.
Case the rule Q-Int-L:
decr(e_1) is an n. It cannot be reduced.
Case the rule Q-Int:
decr(e_1) is an n. It cannot be reduced.
Case the rule Q-Var:
A variable cannot be typed under an empty environment.

□

Lemma 5 (Soundness of Type Inference)
Hypothesis:
(1) $\Gamma \vdash e\colon \rho; C; \mathcal{X}$
(2) $\sigma(C)$ is valid
Conclusion:
$\sigma(\Gamma) \vdash \sigma(e)\colon \sigma(\rho)$

Proof.
Structural induction on e:
Case $e = e_1\ f\ e_2$:
From [1] and Q-Math-Inf (Q-Rel-Inf is similar), we have
(3) $\rho = \gamma\ \alpha$
(4) $\Gamma \vdash e_1\colon (\kappa_1\ \tau_1); C_1$
(5) $\Gamma \vdash e_2\colon (\kappa_2\ \tau_2); C_2$
(6) $C = C_1 \cup C_2 \cup \{\kappa_1 = \kappa_2 = \gamma, \gamma \sqsubseteq l_f, \tau_1 = \tau_2 = Int = \alpha\}$
From [2] and [6], we have
(7) $\sigma(C_1)$ is valid
(8) $\sigma(C_2)$ is valid
From IH on [4], [7], we have
(9) $\sigma(\Gamma) \vdash \sigma(e_1)\colon \sigma(\kappa_1\ \tau_1)$
Similarly, from IH on [5], [8], we have
(10) $\sigma(\Gamma) \vdash \sigma(e_2)\colon \sigma(\kappa_2\ \tau_2)$
From [6], [2], we have
(11) $\sigma(\kappa_1) = \sigma(\kappa_2) = \sigma(\gamma)$
(12) $\sigma(\kappa) \sqsubseteq l_f$
(13) $\sigma(\tau_1) = \sigma(\tau_2) = \sigma(\alpha)$
From [9], [11], [13], we have
(14) $\sigma(\Gamma) \vdash \sigma(e_1)\colon \sigma(\gamma)\ \sigma(\alpha)$
From [10], [11], [13], (and $\alpha = Int$), we have
(15) $\sigma(\Gamma) \vdash \sigma(e_2)\colon \sigma(\gamma)\ \sigma(\alpha)$
From Q-Math, [14], [15], [12], we have

(16) $\sigma(\Gamma) \vdash \sigma(e_1 \; f \; e_2)\colon \sigma(\gamma) \; \sigma(\alpha)$

From [3], [16], we have

$\sigma(\Gamma) \vdash \sigma(e_1 \; f \; e_2)\colon \sigma(\rho)$

Case $e = e_1 \; e_2$:

Similar to the case $e_1 \; f \; e_2$. Application of IH on the subexpressions.

Case $e = \lambda x{:}\rho_1.e'$:

Application of IH to $\Gamma[x \mapsto \rho_1] \vdash e'\colon \rho_2; C$.

Case $e = x$:

Trivial. $\Gamma(x) = \rho$. Thus, $\sigma(\Gamma) = \sigma(\rho)$.

Case $e = n_l$:

Trivial.

Case $e = n$:

Trivial.

□

Lemma 6 (Completeness of Type Inference)

Hypothesis:

(0) $\Gamma \vdash e\colon (\gamma \; \alpha); C; \mathcal{X}$ and

There is a substitution σ such that

(1) $\sigma(\Gamma) \vdash \sigma(e)\colon (\kappa \; \tau)$

(2) $dom(\sigma) \cap \mathcal{X} = \emptyset$

Conclusion:

There is a substitution σ' such that

$\sigma'(C)$ is valid and

$\sigma'(\gamma \; \alpha) = \kappa \; \tau$ and

$\sigma' \setminus \mathcal{X} = \sigma$.

Proof.

Structural induction on e:

Case $e = e_1 \; f \; e_2$:

From the rule Q-Math-Inf (the rule Q-Rel-Inf is similar) and [0], we have

(3) $\Gamma \vdash e_1\colon (\kappa_1 \; \tau_1); C_1; \mathcal{X}_1$

(4) $\Gamma \vdash e_2\colon (\kappa_2 \; \tau_2); C_2; \mathcal{X}_2$

(5) $C = C_1 \cup C_2 \cup \{\kappa_1 = \kappa_2 = \gamma, \gamma \sqsubseteq l_f, \tau_1 = \tau_2 = Int = \alpha\}$

(6) $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \{\gamma, \alpha\}$

(7) $\mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap FV(\kappa_2 \; \tau_2) = \mathcal{X}_2 \cap FV(\kappa_1 \; \tau_1) = \emptyset$

(8) γ, α fresh

From Q-Op, [1], we have

(9) $\sigma(\Gamma) \vdash \sigma(e_1)\colon (\kappa \; \tau)$

(10) $\sigma(\Gamma) \vdash \sigma(e_2)\colon (\kappa\ \tau)$
(11) $\kappa \sqsubseteq l_f$
From [2], [6], we have
(12) $dom(\sigma) \cap \mathcal{X}_1 = \emptyset$
(13) $dom(\sigma) \cap \mathcal{X}_2 = \emptyset$
By IH on [3], [9], [12] we have
There exists σ_1 such that
(14) $\sigma_1(C_1)$ is valid.
(15) $\sigma_1(\kappa_1\ \tau_1) = (\kappa\ \tau)$
(16) $\sigma_1 \setminus \mathcal{X}_1 = \sigma$
By IH on [4], [10], [13] we have
There exists σ_2 such that
(17) $\sigma_2(C_2)$ is valid.
(18) $\sigma_2(\kappa_2\ \tau_2) = (\kappa\ \tau)$
(19) $\sigma_2 \setminus \mathcal{X}_2 = \sigma$
As we have [7] ($\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$), [8], we can define σ' as follows:

$$(20)\ \sigma' = \begin{cases} Y \mapsto U & \text{if } Y \notin \mathcal{X}\ \wedge\ (Y \mapsto U) \in \sigma \\ Y_1 \mapsto U_1 & \text{if } Y_1 \in \mathcal{X}_1\ \wedge\ (Y_1 \mapsto U_1) \in \sigma_1 \\ Y_2 \mapsto U_2 & \text{if } Y_2 \in \mathcal{X}_2\ \wedge\ (Y_2 \mapsto U_2) \in \sigma_2 \\ \gamma \mapsto \kappa \\ \alpha \mapsto \tau \end{cases}$$

From [20], we have
$\sigma'(\gamma\ \alpha) = \kappa\ \tau$
From [20], [2], we have
$\sigma' \setminus \mathcal{X} = \sigma$.
Thus, what is remained to be proved is
$\sigma'(C)$ is valid.
Thus, from [5], we need to prove
$\sigma'(C_1)$ is valid.
$\sigma'(C_2)$ is valid.
$\sigma'(\{\kappa_1 = \kappa_2 = \gamma, \gamma \sqsubseteq l_f, \tau_1 = \tau_2 = \alpha\})$ is valid.
From [20], [8] (γ and α not in C_1), no Y_2 in $\mathcal{X}_2$ is in C_1, [6], [16], [14],
$\sigma'(C_1)$ is valid.
From [20], [8] (γ and α not in C_2), no Y_1 in $\mathcal{X}_1$ is in C_2, [6], [19], [17],
$\sigma'(C_2)$ is valid.
Thus, what is remained to be proved is
$\sigma'(\{\kappa_1 = \kappa_2 = \gamma, \gamma \sqsubseteq l_f, \tau_1 = \tau_2 = \alpha\})$ is valid.
We need to prove
$\sigma'(\gamma) \sqsubseteq l_f$
$\sigma'(\kappa_1) = \sigma'(\gamma)$

$\sigma'(\kappa_2) = \sigma'(\gamma)$
$\sigma'(\tau_1) = \sigma'(\alpha)$
$\sigma'(\tau_2) = \sigma'(\alpha)$
From [20], [11] we have
$\sigma'(\gamma) \sqsubseteq l_f$
From [20], [7] ($\mathcal{X}_2 \cap FV(\kappa_1\ \tau_1) = \emptyset$), [8] ($\{\gamma, \alpha\} \cap FV(\kappa_1\ \tau_1) = \emptyset$) and [16], we have
(21) $\sigma'(\kappa_1) = \sigma_1(\kappa_1)$
From [15], we have
(22) $\sigma_1(\kappa_1) = \kappa$
From (20), we have
(23) $\sigma'(\gamma) = \kappa$
From [21], [22], [23], we have
$\sigma'(\kappa_1) = \sigma'(\gamma)$
The proofs of the remained equations are the same.
$\sigma'(\kappa_2) = \sigma'(\gamma)$
$\sigma'(\tau_1) = \sigma'(\alpha)$
$\sigma'(\tau_2) = \sigma'(\alpha)$

Case $e = e_1\ e_2$:
Similar to the case $e = e_1\ f\ e_2$.
σ' is defined as follows:

$$\sigma' = \begin{cases} Y \mapsto U & \text{if } Y \notin \mathcal{X}\ \wedge\ (Y \mapsto U) \in \sigma \\ Y_1 \mapsto U_1 & \text{if } Y_1 \in \mathcal{X}_1\ \wedge\ (Y_1 \mapsto U_1) \in \sigma_1 \\ Y_2 \mapsto U_2 & \text{if } Y_2 \in \mathcal{X}_2\ \wedge\ (Y_2 \mapsto U_2) \in \sigma_2 \\ \gamma' \mapsto \kappa \\ \alpha \mapsto \tau \end{cases}$$

Case $e = \lambda x{:}\rho_1.e'$:
From the premises of the rule Q-Lam-Inf on $\Gamma \vdash \lambda x{:}\rho_1.e' \colon (\gamma\ \alpha); C'; \mathcal{X}'$, we have
$\Gamma[x \mapsto \rho_1] \vdash e' \colon \rho_2; C; \mathcal{X}$
By the inversion lemma on $\sigma(\Gamma) \vdash \lambda x{:}\sigma(\rho_1).\sigma(e') \colon (\kappa\ \tau)$,
$\kappa = \circ$, $\tau = \sigma(\rho_1) \to \rho_2'$ and $\sigma(\Gamma)[x \mapsto \sigma(\rho_1)] \vdash \sigma(e') \colon \rho_2'$ for some ρ_2',
Thus, IH gives a σ'' such that
$\sigma''(C')$ is valid.
$\sigma''(\rho_2) = \rho_2'$
$\sigma'' \setminus \mathcal{X} = \sigma$
σ' is defined as follows:
$\sigma' = \sigma''[\gamma \mapsto \circ][\alpha \mapsto (\rho_1 \to \rho_2)]$.
The result follows by substitutions.
$\sigma'(\gamma\ \alpha) = \sigma'(\circ)\ \sigma'(\rho_1 \to \rho_2) = \circ\ (\sigma''(\rho_1) \to \sigma''(\rho_2)) =$

$\kappa\ (\sigma(\rho_1) \to \rho_2') = \kappa\ \tau$

Note that $\sigma''(\rho_1) = \sigma(\rho_1)$ because

$\mathcal{X}$ does not contain any type variables of ρ_1 and $\sigma'' \setminus \mathcal{X} = \sigma$

$\sigma'(C')$ is valid because $\sigma''(C)$ is valid and direct substitution.

$\sigma' \setminus \mathcal{X}' = \sigma$ by the definition of σ' and $\sigma'' \setminus \mathcal{X} = \sigma$.

Case $e = x$:

$\sigma' = \sigma$. Trivial.

Case $e = n_l$:

$\sigma' = [\gamma \mapsto l][\alpha \mapsto Int]$. Trivial.

Case $e = n$:

Similar to the case $e = n_l$. Trivial.

□

Lemma 7 (Security)

Proof. Let $A = (A_1, A_2)$ be an adversary attacking program-indistinguishability, with advantage at least ϵ. For each encryption scheme $i = 1, \ldots, n$, we build an adversary (B_1^i, B_2^i) as follows. The algorithm B_1^i takes a public key pk_i and uses the key generation algorithm to generate $(n-1)$ public keys and calls A_1 with the tuple $(pk_1, \ldots, pk_i, \ldots, pk_n)$. It returns the ith component of the output of A_1. Next, when the adversary receives the challenge y_i, she picks $b' \in \{0, 1\}$ at random. Let $b'' = 1 - b'$. She generates a vector $(y_1, \ldots, y_i, \ldots, y_n)$ so that for all components $1 < k < i$, y_k is the encryption of $x_k^{b'}$ and for components $i < l < n$, y_l is the encryption of $x_l^{b''}$. These encryptions are performed using the keys generated in the first step. The calculation of [1] shows that choosing an adversary B^i uniformly at random gives an adversary with advantage ϵ/n. □

3 Constraint Solving

Given an encryption lattice L with elements l and a set of constraints C of the following form, the goal is to find a solution that assigns the largest possible element of L to each qualifier variable. Note that a τ can be a type variable α or a constant type t and a κ can be qualifier variable γ or a qualifier constant q (l or $\circ$).

$$\begin{array}{lll} C ::= & \{\tau_1 = \tau_2\} \ \mid \ \{\kappa_1 = \kappa_2\} \ \mid & \text{Constraint} \\ & \{\gamma \sqsubseteq l\} \ \mid & \\ & C_1 \cup C_2 & \end{array}$$

We first apply unification to the type constraints $\tau_1 = \tau_2$. There is no solution if the unification fails. For the set of qualifier constraints, we find the equivalence classes of variables based on the equal pairs $\kappa_1 = \kappa_2$ in C. If more than one constant is in a class, the constraints are inconsistent and there is no solution. If there is only one constant in a class, we assign the constant to each element of the class. (Note that when the constant $\circ$ is in a class (of qualifier variables), no encryption l can be assigned to the variables of the class and $\circ$ is assigned to them.) For the classes that contain no constant, we proceed as follows. For each class of qualifier variables Q, let L_Q be the set of lattice elements l for which there is a variable $\gamma \in Q$ such that $(\gamma \sqsubseteq l) \in C$. The greatest lower bound of L_Q in L is assigned to each element of Q.

References

[1] O. Baudron, D. Pointcheval, and J. Stern. Extended notions of security for multicast public key cryptosystems. In *ICALP '00*, volume 1853 of *Lecture Notes in Computer Science*, pages 499–511. Springer, 2000.

[2] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in ssh: provably fixing the ssh binary packet protocol. In *CCS '02*, pages 1–11. ACM, 2002.

[3] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2009.

[4] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 578–595. Springer, 2011.

[5] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Computer and Systems Sciences*, 28:270–299, 1984.