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 2
1.1 Source Programs .................................................. 2
1.2 Plaintext Domain .................................................... 3
  1.2.1 Operational Semantics ......................................... 3
1.3 Encrypted Domain .................................................... 3
  1.3.1 Typing .......................................................... 3
  1.3.2 Operational Semantics ......................................... 4
  1.3.3 Type Inference .................................................. 5
1.4 Soundness .......................................................... 6
1.5 Security Guarantees ............................................... 7

2 Proofs 10
2.1 Helper Lemmas .................................................... 10
2.2 Main Lemmas ....................................................... 11

3 Constraint Solving 18
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 $\mathcal{R} = \{<\leq\geq\}$ and we have the set of encryption schemes $L = \{\text{RAND, OP, DET, MH, AH, FHE}\}$. RAND encryption supports no operations, OP supports $\{\leq\geq\}$, DET supports $\{-\}$, MH supports $\{\times\}$, AH supports $\{+\}$, FHE supports all of these operations. We define the encryption lattice on $L$ as follows: $\text{FHE} \sqsubseteq \text{OP}$, $\text{FHE} \sqsubseteq \text{MH}$, $\text{FHE} \sqsubseteq \text{AH}$, $\text{OP} \sqsubseteq \text{DET}$, $\text{DET} \sqsubseteq \text{RAND}$, $\text{MH} \sqsubseteq \text{RAND}$, $\text{AH} \sqsubseteq \text{RAND}$. In this lattice, $\text{FHE}$ is $\bot$ and $\text{RAND}$ is $\top$.

1.1 Source Programs

We define the set of expressions as follows

$$e ::= e_1 e_2 \mid e_1 \ f \ e_2 \mid \text{Program}$$

$$v \mid x$$

$$v ::= \lambda x:\rho.e \mid n \mid n_l \mid \text{Value}$$

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

$$\tau ::= \text{Int} \mid \alpha \mid \rho \to \rho \mid \text{Type}$$

$$\kappa ::= l \mid \circ \mid \gamma \mid \text{Qualifier}$$

$$\rho ::= \kappa \tau \mid \text{Qualified Type}$$

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. ○ is the qualifier for unencrypted data. As any operation is supported on unencrypted data, ⊑ is extended such that ∀ f : ○ ⊑ f

We define \( \text{decr}(e) \) that decrypts the the encrypted numbers in \( e \), as follows: \( \text{decr}(e_1 e_2) = \text{decr}(e_1) \text{decr}(e_2) \), \( \text{decr}(e_1 f e_2) = \text{decr}(e_1) f \text{decr}(e_2) \), \( \text{decr}(\lambda x:\rho.e) = \lambda x:\rho.\text{decr}(e) \), \( \text{decr}(n_1) = n \), \( \text{decr}(n) = n \) and \( \text{decr}(x) = x \).

1.2 Plaintext Domain
1.2.1 Operational Semantics

\[
R ::= \text{[]} \mid R e \mid v R \mid R f e_2 \mid v f R
\]

We characterize the transitions of these operations as follows:

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

**Math-Trans**
\[
n f n' = n'' \quad f \in \mathcal{M} \\
\frac{R[n' f n''] \rightarrow \rho R[n'']}{R[n' f n''] \rightarrow \rho R[n'']}
\]

**Rel-Trans**

\[
n'' = \begin{cases} 
1 & \text{if } (n f n') \\
0 & \text{if } \neg(n f n') 
\end{cases} \\
\frac{R[n' f n''] \rightarrow \rho R[n'']}{R[n' f n''] \rightarrow \rho R[n'']}
\]

1.3 Encrypted Domain
1.3.1 Typing

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

\[
R ::= [ ] \mid R e \mid v R \mid R f e_2 \mid v f R
\]

We characterize the transitions of these operations as follows:

**Q-App-Trans**
\[
R[(\lambda x: \rho_1.e) \nu] \rightarrow R[e[x \mapsto \nu]]
\]

**Q-Math-Trans**
\[
n f n' = n'' \quad f \in \mathcal{M} \\
l \sqsubseteq l_f
\]
\[
R[n_l f n'_l] \rightarrow R[n''_l]
\]

**Q-Rel-Trans**
\[
n'' = \begin{cases} 
1 & \text{if } (n f n') \\
0 & \text{if } -(n f n') 
\end{cases}
\]
\[
f \in \mathcal{R} \\
l \sqsubseteq l_f
\]
\[
R[n_l f n'_l] \rightarrow R[n''_l]
\]
1.3.3 Type Inference

The judgments are of the form $\Gamma \vdash e : \rho; C; X$ i.e. in the type environment $\Gamma$, the expression $e$ has type $\rho$ under the constraints $C$. The set of variables $X$ keeps track of the variables that are introduced in each subderivation.

The set of constraints is defined as

$$C ::= \{\tau_1 = \tau_2\} \mid \{\kappa_1 = \kappa_2\} \mid \{\gamma \subseteq l\} \mid C_1 \cup C_2$$

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**

$$\Gamma[x \mapsto \rho_1] \vdash e : \rho_2; C; X$$

$$C' = C \cup \{\gamma = \alpha, \alpha = \rho_1 \to \rho_2\}$$

$$X' = X \cup \{\gamma, \alpha\} \quad \gamma, \alpha \text{ fresh}$$

$$\Gamma \vdash \lambda x : \rho_1. e : (\gamma \alpha); C'; X'$$

**Q-APP-INF**

$$\Gamma \vdash e_1 : \rho_1; C_1; X_1 \quad \Gamma \vdash e_2 : \rho_2; C_2; X_2$$

$$C = C_1 \cup C_2 \cup \{\rho_1 = \gamma (\rho_2 \to \gamma' \alpha)\}$$

$$X = X_1 \cup X_2 \cup \{\gamma, \gamma', \alpha\}$$

$$X_1 \cap X_2 = X_1 \cap FV(\rho_2) = X_2 \cap FV(\rho_1) = \emptyset$$

$$\gamma, \gamma', \alpha \text{ fresh}$$

$$\Gamma \vdash e_1 \; e_2 : (\gamma' \alpha); C; X$$

**Q-MATH-INF**

$$f \in M$$

$$\Gamma \vdash e_1 : (\kappa_1 \tau_1); C_1; X_1 \quad \Gamma \vdash e_2 : (\kappa_2 \tau_2); C_2; X_2$$

$$C = C_1 \cup C_2 \cup \{\kappa_1 = \kappa_2 = \gamma, \gamma \subseteq I, \tau_1 = \tau_2 = Int = \alpha\}$$

$$X = X_1 \cup X_2 \cup \{\gamma, \alpha\}$$

$$X_1 \cap X_2 = X_1 \cap FV(\kappa_2 \tau_2) = X_2 \cap FV(\kappa_1 \tau_1) = \emptyset$$

$$\gamma, \alpha \text{ fresh}$$

$$\Gamma \vdash (e_1 \; f \; e_2) : (\gamma \alpha); C; X$$
Q-REL-INF

\[ f \in \mathcal{R} \]
\[ \Gamma \vdash e_1 : (\kappa_1 \tau_1) ; C_1 ; \mathcal{X}_1 \quad \Gamma \vdash e_2 : (\kappa_2 \tau_2) ; C_2 ; \mathcal{X}_2 \]
\[ C = C_1 \cup C_2 \cup \{ \kappa_1 = \kappa_2 , \gamma = o , \kappa_1 \sqsubseteq l_f , \tau_1 = \tau_2 = \text{Int} = \alpha \} \]
\[ \mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \{ \gamma , \alpha \} \]
\[ \mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap \text{FV}(\kappa_2 \tau_2) = \mathcal{X}_2 \cap \text{FV}(\kappa_1 \tau_1) = \emptyset \]
\[ \gamma , \alpha \text{ fresh} \]
\[ \Gamma \vdash (e_1 f e_2) : (\gamma \alpha) ; C ; \mathcal{X} \]

Q-VAR-INF

\[ \Gamma(x) = \rho \]
\[ \Gamma \vdash x : \rho ; \emptyset ; \emptyset \]

Q-INT-L-INF

\[ C = \{ \gamma = l , \alpha = \text{Int} \} \quad \mathcal{X} = \{ \gamma , \alpha \} \]
\[ \gamma , \alpha \text{ fresh} \]
\[ \Gamma \vdash n_l : (\gamma \alpha) ; C ; \mathcal{X} \]

Q-INT-INF

\[ C = \{ \gamma = o , \alpha = \text{Int} \} \quad \mathcal{X} = \{ \gamma , \alpha \} \]
\[ \gamma , \alpha \text{ fresh} \]
\[ \Gamma \vdash n : (\gamma \alpha) ; C ; \mathcal{X} \]

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

1.4 Soundness

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

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

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 \( e \rightarrow e' \), then \( \text{decr}(e) \rightarrow_p \text{decr}(e') \).

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

**Lemma 4 (Encryption Domain Completeness).** If \( \emptyset \vdash e_1 : \rho \) and \( e'_1 = \text{decr}(e_1) \) and \( e'_1 \rightarrow_p e'_2 \), then \( e_1 \rightarrow e_2 \) and \( e'_2 = \text{decr}(e_2) \).

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

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

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 \( \sigma \).

**Lemma 6 (Completeness of Type Inference).** If \( \Gamma \vdash e : \rho ; C ; \mathcal{X} \) and
there is a substitution \( \sigma \) such that \( \sigma(\Gamma) \vdash \sigma(e) : \rho' \) and \( \text{dom}(\sigma) \cap \mathcal{X} = \emptyset \)
then there is a substitution \( \sigma' \) 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 ob-
serves 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 \(\lambda\). 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

\[
\text{Adv}_E(A) = \Pr[A_2(y, s) = b] - \frac{1}{2}
\]

where the random variables are distributed uniformly.

An encryption scheme \( E = (K, E, D) \) satisfies single-use *indistinguishability* against chosen plaintext attacks (IND-CPA) if for each adversary \( A \) we have that \( \text{Adv}_E(A) \) is negligible (recall that a function \( f(n) \) is negligible if \( |f(n)| < \frac{1}{\text{poly}(n)} \) 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 \( E \) has an associated indistinguishability property \( \text{IND}(E) \).

In our context, we have a set of inputs \( x_1, \ldots, x_n \) to the program, and use possibly different encryption schemes \( E_1, \ldots, E_n \) for them. We ask, given that each scheme \( E_i \) satisfies \( \text{IND}(E_i) \), 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 component-wise 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 $\rho_2$ with $\Gamma, x: \rho_1 \vdash e : \rho_2$.
- If $\Gamma \vdash n_1 : \rho$ then $\rho = l \text{ int}$.
- If $\Gamma \vdash n : \rho$ then $\rho = \text{int}$.

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 $l \text{ int}$ then $v = n_1$ for some $n$.
- If $v$ is a value of type $\text{int}$ then $v = n$ for some $n$.


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).

- $\text{decr}(e) = \lambda x : \rho_1. e'_1$ then $e = \lambda x : \rho_1. e_1$ where $\text{decr}(e_1) = e'_1$.
- $\text{decr}(e) = n$ then either $e = n$ or $e = n_1$ for some $l$.

Proof. Immediate form the definition of $\text{decr}$.
2.2 Main Lemmas

Lemma 1 (Progress)
Hypothesis:
\[ \emptyset \vdash e : \rho \]
Conclusion:
either \( e \) is a value 
or there is some \( e' \) such that \( e \rightarrow e' \)

Proof.
Induction on the derivation of \( \emptyset \vdash e : \rho \).
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:
\[ \Gamma \vdash e : \rho \]
\[ e \rightarrow e' \]
Conclusion:
\[ \Gamma \vdash e' : \rho \]
Proof.

Straightforward induction on the derivation of $\Gamma \vdash e : \rho$ 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:

Let $e_1$ and $e_2$ be 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 $\nu R$,

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:

$\text{decr}(e) \to_p \text{decr}(e')$

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:
\[ \emptyset \vdash e_1 : \rho, \]
\[ e'_1 = \text{decr}(e_1) \]
\[ e'_1 \rightarrow_\rho e'_2 \]

Conclusion:
\[ e_1 \rightarrow e_2 \]
\[ e'_2 = \text{decr}(e_2) \]

Proof.
Straightforward induction on the derivation of \( \emptyset \vdash e_1 : \rho \).

Case the rule Q-Lam:
\[ \text{decr}(e_1) \] is a lambda term. It cannot be reduced.

Case the rule Q-App:
\[ e_1 = e_{11} e_{12} \]
\[ e'_1 = \text{decr}(e_1) = e'_{11} e'_{12} \] where
\[ e'_{11} = \text{decr}(e_{11}) \] and \( e'_{12} = \text{decr}(e_{12}) \)

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 \( \text{decr}(e) = e' \) and
\[ e_{12} = v \] such that \( \text{decr}(v) = v' \).
Thus, \( e'_2 = e'[x \mapsto v'] = \text{decr}(e)[x \mapsto \text{decr}(v)] \)
By the rule Q-App-Trans, \( e_1 \rightarrow e_2 \) where \( e_2 = e[x \mapsto v] \)
which yields the result.

Otherwise,
Either \( e'_{11} = \text{decr}(e_{11}) \) or \( e'_{12} = \text{decr}(e_{12}) \) 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:
\(\text{decr}(e_1)\) is an \(n\). It cannot be reduced.

Case the rule Q-INT:
\(\text{decr}(e_1)\) is an \(n\). It cannot be reduced.

Case the rule Q-VAR:
A variable cannot be typed under an empty environment.

\[\square\]

**Lemma 5 (Soundness of Type Inference)**

**Hypothesis:**
\begin{enumerate}
  \item \(\Gamma \vdash e : \rho; C; \mathcal{X}\)
  \item \(\sigma(C)\) is valid
\end{enumerate}

**Conclusion:**
\(\sigma(\Gamma) \vdash \sigma(e) : \sigma(\rho)\)

**Proof.**

Structural induction on \(e\):

Case \(e = e_1 \cdot e_2\):
From [1] and Q-MATH-INF (Q-REL-INF is similar), we have
\begin{enumerate}
  \item \(\rho = \gamma \alpha\)
  \item \(\Gamma \vdash e_1 : (\kappa_1 \tau_1); C_1\)
  \item \(\Gamma \vdash e_2 : (\kappa_2 \tau_2); C_2\)
  \item \(C = C_1 \cup C_2 \cup \{\kappa_1 = \kappa_2 = \gamma, \gamma \sqsubseteq l_f, \tau_1 = \tau_2 = \text{Int} = \alpha\}\)
\end{enumerate}

From [2] and [6], we have
\begin{enumerate}
  \item \(\sigma(C_1)\) is valid
  \item \(\sigma(C_2)\) is valid
\end{enumerate}

From IH on [4], [7], we have
\begin{enumerate}
  \item \(\sigma(\Gamma) \vdash \sigma(e_1) : \sigma(\kappa_1 \tau_1)\)
\end{enumerate}

Similarly, from IH on [5], [8], we have
\begin{enumerate}
  \item \(\sigma(\Gamma) \vdash \sigma(e_2) : \sigma(\kappa_2 \tau_2)\)
\end{enumerate}

From [6], [2], we have
\begin{enumerate}
  \item \(\sigma(\kappa_1) = \sigma(\kappa_2) = \sigma(\gamma)\)
  \item \(\sigma(\kappa) \sqsubseteq l_f\)
  \item \(\sigma(\tau_1) = \sigma(\tau_2) = \sigma(\alpha)\)
\end{enumerate}

From [9], [11], [13], we have
\begin{enumerate}
  \item \(\sigma(\Gamma) \vdash \sigma(e_1) : \sigma(\gamma) \sigma(\alpha)\)
\end{enumerate}

From [10], [11], [13], (and \(\alpha = \text{Int}\)), we have
\begin{enumerate}
  \item \(\sigma(\Gamma) \vdash \sigma(e_2) : \sigma(\gamma) \sigma(\alpha)\)
\end{enumerate}

From Q-MATH, [14], [15], [12], we have
From [3], [16], we have

\( \sigma(\Gamma) \vdash \sigma(e_1 f e_2) : \sigma(\gamma) \sigma(\alpha) \)

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 \( \Gamma[x \mapsto \rho_1] \vdash e' : \rho_2 ; C \).

Case \( e = x \):
Trivial. \( \Gamma(x) = \rho \). Thus, \( \sigma(\Gamma) = \sigma(\rho) \).

Case \( e = n \):
Trivial.

Lemma 6 (Completeness of Type Inference)

Hypothesis:
(0) \( \Gamma \vdash e : (\gamma \alpha) ; C \); \( \mathcal{X} \) and
There is a substitution \( \sigma \) such that
(1) \( \sigma(\Gamma) \vdash \sigma(e) : (\kappa \tau) \)
(2) \( \text{dom}(\sigma) \cap \mathcal{X} = \emptyset \)

Conclusion:
There is a substitution \( \sigma' \) 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) \( \Gamma \vdash e_1 : (\kappa_1 \tau_1) ; C_1 ; \mathcal{X}_1 \)
(4) \( \Gamma \vdash e_2 : (\kappa_2 \tau_2) ; C_2 ; \mathcal{X}_2 \)
(5) \( C = C_1 \cup C_2 \cup \{ \kappa_1 = \kappa_2 = \gamma, \gamma \subseteq l_f, \tau_1 = \tau_2 = \text{Int} = \alpha \} \)
(6) \( \mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \{ \gamma, \alpha \} \)
(7) \( \mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap \text{FV}(\kappa_2 \tau_2) = \mathcal{X}_2 \cap \text{FV}(\kappa_1 \tau_1) = \emptyset \)
(8) \( \gamma, \alpha \) fresh
From Q-OP, [1], we have
(9) \( \sigma(\Gamma) \vdash \sigma(e_1) : (\kappa \tau) \)
(10) $\sigma(\Gamma) \vdash \sigma(e_2) : (\kappa \tau)$
(11) $\kappa \subseteq l_f$

From [2], [6], we have

(12) $\text{dom}(\sigma) \cap X_1 = \emptyset$
(13) $\text{dom}(\sigma) \cap X_2 = \emptyset$

By IH on [3], [9], [12] we have

There exists $\sigma_1$ such that

(14) $\sigma_1(C_1)$ is valid.
(15) $\sigma_1(\kappa_1 \tau_1) = (\kappa \tau)$
(16) $\sigma_1 \setminus X_1 = \sigma$

By IH on [4], [10], [13] we have

There exists $\sigma_2$ such that

(17) $\sigma_2(C_2)$ is valid.
(18) $\sigma_2(\kappa_2 \tau_2) = (\kappa \tau)$
(19) $\sigma_2 \setminus X_2 = \sigma$

As we have [7] ($X_1 \cap X_2 = \emptyset$), [8], we can define $\sigma'$ as follows:

(20) $\sigma' = \begin{cases} 
Y \mapsto U & \text{if } Y \notin X \land (Y \mapsto U) \in \sigma \\
Y_1 \mapsto U_1 & \text{if } Y_1 \in X_1 \land (Y_1 \mapsto U_1) \in \sigma_1 \\
Y_2 \mapsto U_2 & \text{if } Y_2 \in X_2 \land (Y_2 \mapsto U_2) \in \sigma_2 \\
\gamma \mapsto \kappa \\
\alpha \mapsto \tau
\end{cases}$

From [20], we have

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

From [20], [2], we have

$\sigma' \setminus X = \sigma$.

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.
$\sigma'\{\kappa_1 = \kappa_2 = \gamma, \gamma \subseteq l_f, \tau_1 = \tau_2 = \alpha\}$ is valid.

From [20], [8] ($\gamma$ and $\alpha$ not in $C_1$), no $Y_2$ in $X_2$ is in $C_1$, [6], [16], [14],
$\sigma'(C_1)$ is valid.

From [20], [8] ($\gamma$ and $\alpha$ not in $C_2$), no $Y_1$ in $X_1$ is in $C_2$, [6], [19], [17],
$\sigma'(C_2)$ is valid.

Thus, what is remained to be proved is

$\sigma'\{\kappa_1 = \kappa_2 = \gamma, \gamma \subseteq l_f, \tau_1 = \tau_2 = \alpha\}$ is valid.

We need to prove

$\sigma'(\gamma) \subseteq l_f$
$\sigma'(\kappa_1) = \sigma'(\gamma)$

16
\[ \sigma'(\kappa_2) = \sigma'(\gamma) \]
\[ \sigma'(\tau_1) = \sigma'(\alpha) \]
\[ \sigma'(\tau_2) = \sigma'(\alpha) \]

From [20], [11] we have
\[ \sigma'(\gamma) \sqsubseteq \lf \]

From [20], [7] \((X_2 \cap FV(\kappa_1 \tau_1) = \emptyset)\), [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 \( e = e_1 \ f \ e_2 \).

\( \sigma' \) is defined as follows:
\[ \sigma' = \begin{cases} 
Y \mapsto U & \text{if } Y \notin X \land (Y \mapsto U) \in \sigma \\
Y_1 \mapsto U_1 & \text{if } Y_1 \in X_1 \land (Y_1 \mapsto U_1) \in \sigma_1 \\
Y_2 \mapsto U_2 & \text{if } Y_2 \in X_2 \land (Y_2 \mapsto U_2) \in \sigma_2 \\
\gamma' \mapsto \kappa & \text{if } \gamma' \mapsto \kappa \\
\alpha \mapsto \tau & \text{if } \alpha \mapsto \tau 
\end{cases} \]

Case \( e = \lambda x : p_1 . e' \):

From the premises of the rule Q-LAM-INF on
\[ \Gamma \vdash \lambda x : p_1 . e' : (\gamma, \alpha) ; C' ; X' \], we have
\[ \Gamma[ x \mapsto p_1 ] \vdash e' : \rho_2 ; C ; X \]

By the inversion lemma on \( \sigma(\Gamma) \vdash \lambda x : \sigma(p_1) . e' : (\kappa, \tau) \),
\[ \kappa = \circ, \tau = \sigma(p_1) \rightarrow \rho_2 \] and \( \sigma(\Gamma)[ x \mapsto \sigma(p_1) ] \vdash e' : \rho_2 \) for some \( \rho_2 \).

Thus, IH gives a \( \sigma'' \) such that
\[ \sigma''(C') \text{ is valid.} \]
\[ \sigma''(\rho_2) = \rho_2' \]
\[ \sigma'' \setminus X = \sigma \]

\( \sigma' \) is defined as follows:
\[ \sigma' = \sigma''[\gamma \mapsto \circ[\alpha \mapsto (p_1 \rightarrow p_2)]] \]

The result follows by substitutions.
\[ \sigma'(\gamma, \alpha) = \sigma'(\circ) \ \sigma'(\rho_1 \rightarrow \rho_2) = \circ(\sigma''(\rho_1) \rightarrow \sigma''(\rho_2)) = \]
\( \kappa (\sigma (\rho_1) \rightarrow \rho'_2) = \kappa \tau \)

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

\( \mathcal{X} \) does not contain any type variables of \( \rho_1 \) 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 \( \sigma' \) and \( \sigma'' \setminus \mathcal{X} = \sigma \).

Case \( e = x \):

\( \sigma' = \sigma \). Trivial.

Case \( e = n_l \):

\( \sigma' = [\gamma \mapsto l][\alpha \mapsto \text{Int}] \). Trivial.

Case \( e = n \):

Similar to the case \( e = n_l \). Trivial.

\end{proof}

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 \( \tau \) can be a type variable \( \alpha \) or a constant type \( t \) and a \( \kappa \) can be qualifier variable \( \gamma \) or a qualifier constant \( q \) (\( l \) or \( \circ \)).

\[
C ::= \{ \tau_1 = \tau_2 \} \mid \{ \kappa_1 = \kappa_2 \} \mid \{ \gamma \subseteq l \} \mid C_1 \cup C_2
\]

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 $(\gamma \subseteq l) \in C$. The greatest lower bound of $L_Q$ in $L$ is assigned to each element of $Q$. 
References


