Automorphisms of Types and Cryptography

PCA-22, Saint-Petersburg

Sergei Soloviev, IRIT, France
(partly based on joint work with Jan Malakhovski [6])

06/05/2022

Starting Point

- Starting point of this reflection is that we may use morphisms (transformations) of types in cryptography if one wants **to encrypt structured objects of a type** instead of mere “texts”.
- **Example.** Consider the type

  $$P = (A \to A) \to \dots \to (A \to A) \to (A \to A).$$

  The elements of “premises” are functions $f_i : A \to A$, and “conclusion” as well. So an element of P is, e.g., the composition operator. In encrypted form the order of f_i may be changed.

Isos, Autos and Types

- In Type Theory one may introduce categorical structure taking types as objects and closed terms $t : A \to B$ as morphisms (up to usual equivalence relation based on normalization).
- Identity Id_A is represented by $\lambda x : A.x : A \to A$.
- As usual, $t : A \to B, t^{-1} : B \to A$ are mutually inverse isomorphisms if $t^{-1} \circ t \equiv id_A$ and $t \circ t^{-1} \equiv id_B$.
- E.g., $\lambda z : B_1 \to (B_2 \to C).\lambda x_2 : B_2.\lambda x_1 : B_1.(zx_1x_2)$ is an isomorphism from $B_1 \to (B_2 \to C)$ to $B_2 \to (B_1 \to C)$
- It works for different systems of Type Theory and λ-calculus.
- An automorphism is an isomorphism $t : A \to A$.
- Non-trivial automorphism
  $\lambda z : B \to (B \to C).\lambda x_2 : B.\lambda x_1 : B.(zx_1x_2)$
- I.e., we take $B_1 = B_2$ above.

Isos, Autos and Types

- As usual, $t : A \to B, t^{-1} : B \to A$ are mutually inverse isomorphisms if $t^{-1} \circ t \equiv id_A$ and $t \circ t^{-1} \equiv id_B$.
- E.g., $\lambda z : B_1 \to (B_2 \to C).\lambda x_2 : B_2.\lambda x_1 : B_1.(z x_1 x_2)$ is an isomorphism from $B_1 \to (B_2 \to C)$ to $B_2 \to (B_1 \to C)$
- It works for different systems of Type Theory and λ-calculus.
- An automorphism is an isomorphism $t : A \to A$.
- Non-trivial automorphism
  $\lambda z : B \to (B \to C).\lambda x_2 : B.\lambda x_1 : B.(z x_1 x_2)$
- I.e., $B_1 = B_2$ above.

Isos, Autos and Types

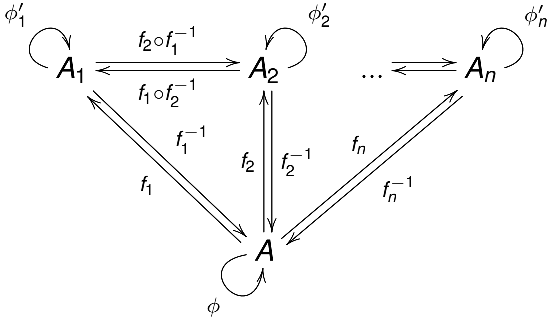
- For each type A the automorphisms $A \rightarrow A$ form the group of automorphisms $Aut(A)$.
- It may be seen as subcategory of the groupoid of isomorphisms .
- i.e., the subcategory of the category of types and deductions - or terms - with the same objects and only isomorphisms as morphisms.
- If we fix a type, we may consider also the groupoid $Gr(A)$ of types isomorphic to A.

Isos, Autos and Types

- To distinguish: the isomorphism relation $\sim$ and the isomorphisms as morphisms.
- $A \sim B$ iff $\exists t : A \rightarrow B.(t\ is\ an\ iso)$.
- The equivalence class w.r.t $\sim$ of A may contain one element while $Aut(A)$ is non trivial: e.g., for $X \rightarrow (X \rightarrow X)$ (X is a type variable).
- Also, $Aut(A)$ may be trivial ($Aut(A) = \{id\}$) while the $\sim$-equivalence class is non-trivial: e.g., for $X \rightarrow (Y \rightarrow Z)$.

Isos, Autos and Types

In type theories below the number of types isomorphic to A is finite, so the groupoid may be represented by the diagram



$$\begin{tikzcd}
A_1 \arrow[r, shift left, "f_2\circ f_1^{-1}"] \arrow[loop left, "\phi'_1"] \arrow[dr, shift left, "f_1^{-1}"] & A_2 \arrow[l, shift left, "f_1\circ f_2^{-1}"] \arrow[loop right, "\phi'_2"] \arrow[d, shift left, "f_2^{-1}"] & \dots \arrow[r, shift left] & A_n \arrow[l, shift left] \arrow[loop right, "\phi'_n"] \arrow[dll, shift left, "f_n^{-1}"] \\
& A \arrow[ul, shift left, "f_1"] \arrow[u, shift left, "f_2"] \arrow[urr, shift left, "f_n"] \arrow[loop below, "\phi"] & &
\end{tikzcd}$$

where $A_1, ..., A_n$ are all types that are isomorphic (but not equal) to A (and to each other), f_i, f_i^{-1} denote the fixed isomorphisms and their inverses, ϕ denotes an arbitrary automorphism of A and $\phi'_i = f_i \circ \phi \circ f_i^{-1}$ $(1 \leq i \leq n)$.

Automorphism Groups

- **Second order λ-calculus without constants.**
- The types are now considered up to renaming of bound variables.
- Simple types are included, and $\overline{\forall}.A$ denotes the universal closure of any type A (quantification over all free variables).

Theorem

For every finite group G there exists a type A_G in simply typed λ-calculus such that the group $Aut(\overline{\forall}.A_G)$ is isomorphic to G.

- **Remark.** The question about optimal presentation may be considered separately.

Automorphism groups

- In the case with **dependent product** (e.g., Z. Luo's typed logical framework *LF*) the universal quantification is modelled by dependent product.
- Instead of $\forall X.B(X)$ we write $(X : \mathit{Type})B(X)$. We can obtain a **similar theorem**: any finite group may be represented in *LF*.
- The case of simply typed λ-calculus is more limited.

Theorem

The groups $Aut(A)$ for simple types A are (up to isomorphism of groups) exactly the groups that may be obtained from symmetric groups by cartesian product and wreath product.

- By an old Jordan's theorem: exactly the groups of automorphisms of finite trees. (Not C_3 for example.)

Applications to Security

- **What may be the use** of λ-terms, isomorphisms and automorphisms in a security-oriented picture?
- **A general idea**: lambda terms may be seen as derived combinators. They may be used to create any program from more elementary building blocks of code.
- To use them “intelligently” to hide (and thus protect) the way how the main program is built from these elementary blocks; if necessary, even the specification may be hidden.
- Isos and autos to encrypt and decrypt (invertibility).

Applications to Security

- Consider some type S. The closed terms $F : S$ represent combinators that may take other terms as arguments.
- Let $f_{1 \div n}$ abbreviate $f_1, ..., f_n$.
- If we take $F \equiv \lambda f_{1 \div n} : X \to X \lambda x : X.(f_{\sigma(1)}(...(f_{\sigma(n)} x)...)$ (σ a permutation of $\{1, ..., n\}$), and apply to some concrete $\phi_{1 \div n}$, it will combine them in any desired order. The $\phi_{1 \div n}$ themselves may be even coding functions.
- If $F \equiv \lambda f_{1 \div n} : X \to X \lambda x : X.f_i x$ then one of ϕ will be selected, etc.
- If $\Phi \equiv \lambda G : S.G : S \to S$, then we may first apply Φ to some operator, like F above, and then "feed" $\phi' s$ (and x in the end).

Applications to Security

- In this example $\lambda G : S.G$ belongs to $\lambda^1\beta\eta$.
- In $\lambda^2\beta\eta$ we may add a second-order λ and consider $\Theta \equiv \lambda X.\lambda G : S.G$.
- In this way the type X also becomes one of controlled parameters, for example it may be *Nat*, *Bool* or any other type.
- If dependent types are admitted, the type X itself may depend on terms as parameters.
- **Remark 1.** If we want to use terms (isos, autos) to encrypt, we encrypt **programs**, not texts, and **all remains well-typed**.

ElGamal with Autos

- Illustration: **ElGamal** cryptosystem (cf. [3, 4]).
- The protocol may use the iterations of a distinguished automorphism $g : A \to A$, where g^m is $g \circ \ldots \circ g$ (m times).
- **Private Key:** $m, m \in N$. **Public Key:** g and g^m.
  Encryption. To send a message $a : A$ (in our approach it is not a plain text, but an element of type A, and may have more complex structure) Bob computes g^r and g^{mr} for a random $r \in N$. The ciphertext is $(g^r, g^{mr}a)$.
  Decryption. Alice knows m, so if she receives the ciphertext $(g^r, g^{mr}a)$, she computes g^{mr} from g^r, then $(g^{mr})^{-1}$, and then computes a from $g^{mr}a$.

ElGamal with Autos

- **Remark.**We do not consider here the cryptosystems like **MOR** based on a more sophisticated group theory [4] but they, too, can be represented in type theory using the results of [5].
- By encoding a finite cyclic group of prime order as a group of automorphism of some type we can implement ElGamal (or any other cryptographic protocol based on finite groups) since the composition and inverse of type automorphisms (represented by finite hereditary permutations [1]) can be computed in linear time.

ElGamal with Autos

- However, finite cyclic groups are not the only possibility.
- Let us recall that the longest period in the symmetric group S_n is given by Landau function $\sim n^{\sqrt{n}}$.
- It corresponds to $Aut(a \to ... \to a \to p)$. This gives an idea of the length of the periods of automorphisms $f \in Aut(A)$.
- This means that the maximal period of an element in the group $Aut(A)$ may be quite high.

ElGamal with Autos

- Clearly, this scheme is less efficient than ElGamal over long integers. However, it has several possible advantages.
- Selecting “interesting” groups makes ElGamal computations “interestingly” inefficient (requiring non-trivial reductions to compute).
- This can be used for proof-of-work algorithms resistant to most conventional computational models (ASICs, GPUs, CPUs).
- This scheme also preserves the structure of $a : A$. (Which, among other things, means that Alice needs not typecheck the decrypted a if she trusts Bob to typecheck his.)

Other Possibilities

- The type S of the combinator F may have many automorphisms which form a subset of all possible isomorphisms to/from this type.
- Automorphisms, in difference from isomorphisms, do not change the types of parameters (taken in a fixed order) that F can be applied to. So, if an automorphism θ of S is applied to F, the application $\theta(F)$ to $t_{1 \div n}$ is valid iff the application $Ft_{1 \div n}$ is valid.
- In difference from automorphisms, an action of an isomorphism θ' (which is not an automorphism) may make invalid the application $\theta'(F)t_{1 \div n}$.
- This fact may be exploited to detect code transformations performed by third-parties or to execute **zero-knowledge proof** protocols if the distinctions between some type variables remain hidden from external observers.

Relative Sizes

- Size of a $\sim$-equivalence class (recurrent formula).
- Let $A = A_1 \to \ldots \to A_n \to p$.
- Some A_i may be isomorphic, some not, let there be “subclasses” (intersections with $A_1, \ldots, A_n$) with k_1,..., k_m elements; $n = k_1 + \ldots + k_m$.
- Let n_i be the full size of the i-th class.
- The size of the $\sim$-equivalence class of A is

$$\frac{n!}{k_1! \ldots k_m!} n_1^{k_1} \ldots n_m^{k_m}.$$

Relative Sizes

For each $A^{(i)} \sim A$ we may fix an isomorphism $f_i : A \to A^{(i)}$

$$\begin{tikzcd}
A^{(1)} & \dots & A^{(i)} \\
\dots & A \arrow[ul, "f_1"'] \arrow[ur, "f_i"] \arrow[dl, "f_k"] \arrow[dr, "f_j"'] & \dots \\
A^{(k)} & \dots & A^{(j)}
\end{tikzcd}$$

and any other isomorphism $A \to A^{(i)}$ may be obtained as composition with some automorphism $A \to A$. As consequence, the number of isomorphisms $A \to \dots$ is given by

$$|Aut(A)| \cdot \frac{n!}{k_1! \dots k_m!} n_1^{k_1} \dots n_m^{k_m}$$

Erasures and Finite Hereditary Permutations

- There is a fundamental theorem by Dezani-Ciancaglini that $\beta\eta$-invertible terms in the untyped λ-calculus are exactly the finite hereditary permutations (f.h.p.).
- Based on this theorem, it is possible to show that $t : A \to B$ is an isomorphism iff its *erasure* is an f.h.p.
- Similar fact was established by Soloviev in [5] for Luo's LF with dependent product.
- This property has still to be explored, however, in the security context.

Normal versus Non-Normal

- **This approach is not yet "ready to use".** One of the difficulties: for a type isomorphism/automorphism in *normal form* it is easy to compute the inverse. And this is not good for cryptography.
- **To be explored:** why then the terms used for encoding have to be in normal form? Normalization of a term may give "size explosion". And reduction sequence may be quite long (more that exponential). So it would be good if applying the Encryption and Decryption we might avoid normalization.

Normal versus Non-Normal

- **Example.** (Not directly related to isomorphisms.) It is well known that natural numbers may be represented by so called *Church numerals*. In their typed version (for a type A) the number n is represented by $\lambda x : A \to A.\lambda y : A.(x(...(xy)...)$ (x is repeated n times).
- This is obviously related to “unary representation” of natural numbers. The term above is normal. One may define arithmetical operations and other arithmetical functions by application of other λ-terms to numerals. (In fact, using untyped lambda calculus and corresponding version of numerals, any partial recursive function may be represented.)

Normal versus Non-Normal

- However, one may define also λ-terms representing binary notation. It is enough to apply (in appropriate order) two terms t_0 and t_1 representing multiplication by 2 and multiplication by 2 plus 1, to the term representing 0.
- Curious observation is that binary numbers are thus represented by terms that are not normal. If we normalize them we obtain exponentially longer presentation by standard Church numerals. However, to execute arithmetical operations we do not need to normalize these terms. The familiar algorithms for binary numbers may be adjusted.

Conclusion

- Many questions ought to be solved to make practical the ideas outlined in this paper. The precise communications protocols should be elaborated that would make use of the distinction between iso- and automorphisms, public and private type information.
- The complexity of algorithms (for example, for reconstruction of typed isomorphisms from erasure) needs to be investigated much more precisely.
- Still, we believe that the use of type theory and λ-calculus as a higher-level formal language for data protection (especially software protection) and detection of attacks has good perspectives and should be developed further.

THANKS FOR YOUR ATTENTION!

References

- Di Cosmo, R. (1995) *Isomorphisms of types: from lambda-calculus to information retrieval and language design.* Birkhauser.
- Heather, J., Lowe, G., and Schneider, S. (2003) How to prevent type flaw attacks on security protocols. J. of Computer Security, 11(2), 217-244.
- Hoffstein, J., Pipher, J. and Silverman, J.H. (2008) An introduction to mathematical cryptography. Springer, New York, 2008.
- Mahalanobis, A. (2015) The MOR cryptosystem and finite p-groups. Contemp. Math., **633**, 81-95.
- Soloviev, S. (2019) Automorphisms of Types in Certain Type Theories and Representation of Finite Groups. *Math. Structures in Computer Science*, 29(4): 511-551.
- Soloviev, S., Malakhovski, J. (2018) Automorphisms of Types and Their Applications. (Engl.) Zap. N. Semin. POMI, 468: 287-308.