Narrowing Terminates for Encryption

Jonathan K. Millen
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730

Hai-Ping Ko*
GTE Government Systems Corporation
77 "A" Street
Needham, MA 02194

Abstract

Many techniques for protocol analysis use term replacement rules to express the reduction properties of symbolic encryption operations. Some approaches must solve equations in those operators, using sequences of narrowing steps. It is shown that every infinite sequence of narrowing steps for popular abstract encryption operators has a loop, and hence there is a terminating algorithm to solve such equations by searching all sequences of narrowing steps.

1 Introduction

Current techniques for security analysis of authentication and key distribution protocols represent encryption operators symbolically and specify their properties with abstract rules or axioms. Approaches that employ state-transition models, such as the Prolog state-search systems [3,4], those that use general purpose specification and verification systems [2], temporal logic [1], model checking [5], and others, typically use explicit or implicit term replacement rules to express the properties of these operators. An example of a term replacement rule is

$$d(K, e(K, X)) \rightarrow X,$$

expressing the property that an encryption in a symmetric-key system is reversed by a decryption with the same key. (We use capital letters for pattern variables that can be instantiated.) Encryption reverses decryption similarly.

Systems that search backwards from an insecure goal state to construct attacks, such as the Interrogator [4] and the NRL tool [3], have to solve equations when attempting to instantiate a state transition rule. An example of an equation is

$$d(k, X) = a,$$

which can be solved by making the substitution $X \rightarrow e(k, a)$ and then applying the rule to reduce the equation to an identity. (Not all equations are this easy; as an exercise, try $e(X, e(e(Y, Z), d(Y, d(X, X)))) = X$.)

The only general approaches available for solving equations in the context of a set of term replacement rules involve some form of narrowing. Meadows was the first to implement a narrowing algorithm for protocol analysis [3]. The term "narrowing" is used both to describe an elementary step in a solution process and an equation-solving algorithm that uses narrowing steps.

A challenge in the design of narrowing algorithms is proving termination, since it is possible to have an infinite sequence of narrowing steps, whether or not a solution exists for the equation. Justifications of various narrowing algorithms typically show completeness rather than termination. Non-deterministic ("oracle") algorithms, which can be implemented with a depth-first search, terminate when a solution exists but might not terminate when a solution does not exist.

A rule set can be well-behaved in the sense that reductions always terminate uniquely, yet infinite narrowing sequences exist. For example, consider the term replacement rule

$$f(g(X)) \rightarrow h(f(X)).$$

The system with just this one rule reduces any expression to a canonical form in a finite number of steps (bounded by the number of gs occurring in it). However, because narrowing permits substitution before reduction, we can have substitutions like $X \rightarrow g(X')$ and we can generate the following infinite narrowing sequence:

$$h(f(X)) \rightarrow h(f(g(X'))) \rightarrow h(h(f(X'))) \rightarrow ...$$

The purpose of this paper is to show that those of us using symbolic encryption rules are in luck. It is still possible to have infinite narrowing sequences, e.g.

[1]Work performed while at The MITRE Corporation

$$h(X, d(K, X)) \rightarrow h(e(K, X'), d(K, e(K, X'))$$
$$\rightarrow h(e(K, X'), X')$$
$$\rightarrow h(e(K, d(K, X'')), d(K, X''))$$
$$\rightarrow h(X'', d(K, X'')) \rightarrow ...$$

but the sequence gets into a loop, since it periodically repeats earlier expressions (up to a renaming of variables). We will show that this always occurs for certain types of popular abstract encryption rules. This means that one can, in principle, construct a narrowing algorithm for these rules that defends itself against nontermination by checking for loops.

The rules for which we can show termination are those of the form

$$f(\alpha, g(\beta, Y)) \rightarrow Y$$

for any operators f and g, and where α and β are either (1) both constants, or (2) the same variable.

Rules of type (1) apply to public-key systems, where α and β form a public/private key pair (in either order). Rules of type (2) apply to symmetric-key systems.

2 Background

2.1 Definitions

We begin with a finite set $\mathcal{C}$ of *constants* and a finite disjoint set $\mathcal{P}$ of *operators*. There is also an infinite set $\mathcal{V}$ of *variables*. From these we make terms.

Definition 2.1 (Term) *A term is either*

a variable,
a constant, or
a form $e(t_1, \ldots, t_n)$

where e is an operator and $t_1, \ldots, t_n$ are terms.

Notational convention. Any term represented by a capital letter is assumed to be a variable. Thus, terms X and Y are variables, and terms t and a may be constants, variables, or operator forms.

Definition 2.2 (Replacement) *If $t, a,$ and b are terms, $t[a \rightarrow b]$ is defined recursively by*

if $t = a$ *then* $t[a \rightarrow b] = b$;
else if $t = e(t_1, ..., t_n)$
then $t[a \rightarrow b] = e(t_1[a \rightarrow b], ..., t_n[a \rightarrow b])$
else $t[a \rightarrow b] = t$.

The function taking t to $t[a \rightarrow b]$ will be referred to as a replacement, and denoted $a \rightarrow b$.

Definition 2.3 (Indexing) *If $t = e(t_1, ..., t_n)$ and $1 \leq i \leq n$ then $t[i] = t_i$; otherwise $t[i]$ is undefined. In this context i is called an index.*

If u is a (possibly null) string $i_1 i_2 ... i_n$ of indices, $t[u]$ is defined recursively as follows:

$t[\lambda] = t$ *(where λ is the null string);*
$t[iu] = t[i][u]$ *if i is an index and u is an index string.*

An index string is called an address.

Definition 2.4 (Subterm) *a is a proper subterm of b, written $a < b$, if $a = b[u]$ for some non-null address u. Also, a is a subterm of b if $a < b$ or $a = b$.*

Definition 2.5 (Term Interior) $\bar{t} = \{a | a < t\}$

2.2 Narrowing

Definition 2.6 (Substitution) *A substitution σ is a function on a finite set of variables into the set of terms, such that no element of the domain of σ is a subterm of any term in the range of σ.*

That is, if σ is defined on $\{X_1, ..., X_n\}$, no X_i is a subterm of any $\sigma(X_j)$. We may write $\sigma = [X_1 \rightarrow x_1]...[X_n \rightarrow x_n]$ where $x_i = \sigma(X_i)$, and define

$$t\sigma = t[X_1 \rightarrow x_1]...[X_n \rightarrow x_n].$$

Note that the order in which the substitutions are applied is immaterial. Strictly speaking, what we have defined is an *idempotent* substitution. Idempotence is important because non-idempotent substitutions do not serve as solutions to equations. For example, the non-idempotent substitution $X \rightarrow f(X)$ does not solve $X = f(X)$.

The composition of idempotent substitutions is not necessarily idempotent. $A \rightarrow B$ is idempotent and so is $B \rightarrow f(A)$, but the composition takes A to $f(A)$. Variables in the range of a substitution should not occur in the domain of any substitution previously applied. The easy way to retain idempotence is to use only substitutions whose images have only new variables that have not been used before.

A *reduction* is just a replacement which is an instantiation of, and hence justified by, a rule. A narrowing step is a substitution followed by a reduction. The purpose of the substitution is to unify the term with an instance of the rule. A narrowing step differs from a reduction in that some variable in the term is instantiated. Thus, one uses narrowing to solve equations, but only reductions may be used to simplify terms. In a narrowing step, both the substitution and the reduction are non-vacuous.

The definition given here is not as restrictive as the one in [6], which says that the substitution is the most general one that makes the replacement possible. A "most general" unifying substitution (MGU) is one that cannot be obtained by instantiating any other unifying substitution. For unifiable terms, most-general unifiers always exist and are unique up to a variable renaming.

The most-general unifier does not necessarily introduce new variables, and hence might not preserve idempotence. What we want is the following:

Definition 2.7 (Almost-General Unifier (AGU)) *A substitution that unifies terms s and t is almost-general if it can be expressed as $\sigma\tau$ where σ is a most-general unifier of s and t and τ maps variables occurring in the range of σ to new variables.*

Narrowing is also usually defined in the context of a set of reduction rules; we will furnish that context later by confining ourselves to encryption rules.

Definition 2.8 (Narrowing Step) *$t[\sigma, b \to y]t'$ is a narrowing step if:*

1. *σ is a substitution $[X_1 \to x_1]...[X_n \to x_n]$,*
2. *for each i, $x_i < b$,*
3. *for some u, $t\sigma[u] = b$, where σ is an AGU of $t\sigma[u]$ and b,*
4. *$t' = t\sigma[b \to y]$,*
5. *b is not a constant or variable.*

In this context u is said to be a *narrowing site.*

The requirement that $x_i < b$ prevents a degenerate narrowing step that first replaces X by b (or something containing b) and then reduces b back to y. One could have accomplished the same thing by a substitution.

2.3 Elementary Properties

Here are some useful elementary properties of addressing. The proofs are straightforward and will be omitted.

Lemma 2.1 (Index After Replacement) *If $t[a \to b][i]$ is defined for an index i and $t \neq a$ then $t[a \to b][i] = t[i][a \to b]$.*

Definition 2.9 (Reaches) *u reaches a in t if $u = vw$ and $t[v] = a$.*

Note that if u reaches a in t, $t[u]$ is not necessarily defined. It may be that $u = vw$ where $t[v] = a$ and $a[w]$ is undefined, as for example if a is a variable and w is non-null.

Observe also that an address u can reach at most one variable, in which case $t[u]$ is that variable, if it is defined.

Lemma 2.2 (Address After Replacement) *Suppose that $t[a \to b][u]$ is defined. Then:*

1. *if u reaches a in t then $t[a \to b][u] = b[w]$ where $u = vw$ and $t[v] = a$.*
2. *if u does not reach a in t then $t[a \to b][u] = t[u][a \to b]$.*

Corollary 2.1 (Address After Substitution) *Suppose that $t\sigma[u]$ is defined. Then:*

1. *if u reaches X in t and $\sigma(X) = x$ then $t\sigma[u] = x[w]$ where $u = vw$ and $t[v] = X$.*
2. *if u does not reach any X in t in the domain of σ, then $t\sigma[u] = t[u]\sigma$.*

3 Narrowing With Encryption Rules

3.1 Encryption Rules

Definition 3.1 (Encryption Rules) *Encryption rules are replacements of the form*

$$f(\alpha, g(\beta, Y)) \to Y$$

for any operators f and g, and where α and β are either (1) both constants, or (2) the same variable.

When an encryption rule is applied in a narrowing step, there are two ways it can happen, depending on the form of the subterm at a narrowing site.

3.2 Type C Steps

If the subterm at a narrowing site has the form $f(a, g(b, c))$, the narrowing step is said to be of *type C*.

In a type C narrowing step, the rule is applied by unifying a and b with α and β respectively. We will show that since the unifications are almost-general ones, a type C narrowing step reduces the number of variables occurring in a term, and hence only a finite number of consecutive type C steps can occur.

Definition 3.2 (Type C Narrowing Step) *A narrowing step $t[\sigma, f(r, g(s, y)) \to y]t'$ is of type C if there exists u such that*

1. *$t[u] = f(p, g(q, z))$ for some p, q, z*

2. *$t[u]\sigma = f(r, g(s, y))$.*

Note that the replacement in the narrowing step is an instantiation of a rule rather than the original rule. Thus the right side y may be a form or constant rather than a variable. The instantiation substitutes values for the pattern variables in the rule so that the left side of the rule becomes identical to a subterm of the term to which the narrowing step is applied.

Proposition 3.1 (Fewer Variables After MGU) *Given terms $s < r$ and $t < r$ with $s \neq t$, if σ is most-general such that $s\sigma = t\sigma$, then $r\sigma$ has fewer variables than r.*

Proof. Let $Vars(r)$ be the set of variables of r. It is known (see [6]) that there is a most-general unifier σ' of s and t such that all variables in the range of σ' are variables of s or t. Hence $Vars(r\sigma') \subseteq Vars(r)$. However, any non-empty substitution eliminates the variables in its domain, and σ' must be non-empty because $s \neq t$. Hence $Vars(r\sigma')$ is a proper subset of $Vars(r)$. Since σ' and σ are both most-general, σ is a renaming of σ', and $Vars(r\sigma)$ is the same cardinality as $Vars(r\sigma')$. ■

Corollary 3.1 (Fewer Variables After Type C) *If $t[\sigma, f(r, g(s, y)) \rightarrow y]t'$ is of type C, then t' has fewer variables than t.*

Proof. Write the AGU σ as $\rho\tau$ where ρ is an MGU and τ renames the variables occurring in the range of ρ to new variables. Apply the previous proposition, noting that the purpose of the substitution is to perform unification, so if no non-trivial unification is necessary, the step is a reduction rather than a narrowing. Finally, observe that after ρ reduces the number of variables, τ does not change the number of variables. ■

3.3 Type V Steps

If the subterm at a narrowing site has the form $f(q, X)$, the narrowing step is said to be of *type V*.

In a type V narrowing step, the rule is applied by making a substitution $X \rightarrow g(s, Y)$, where s is some term that unifies with β, and Y is a new variable (one that does not occur in the term being narrowed). Any other choice in place of Y would make the substitution less general. The newness of Y also implies that a type V step cannot also be a type C step.

We will show that this type of step cannot be applied an infinite number of times without looping.

Definition 3.3 (Type V Narrowing Step) *A narrowing step $t[\sigma, f(r, g(s, Y)) \rightarrow Y]t'$ is of type V if there exists u such that*

1. *$t[u] = f(q, X)$ for some q, X,*

2. *$t[u]\sigma = f(r, g(s, Y))$,*

3. *$Y \not< t$.*

Our proof of termination requires a specific property stated in the lemma below, valid for type V steps arising from encryption rules. We will also show, in the same lemma, that type V steps do not increase the number of variables in a term.

Lemma 3.1 (Type V With Encryption) *If $t[\sigma, f(r, g(s, Y)) \rightarrow Y]t'$ is type V, and $f(r, g(s, Y)) \rightarrow Y$ is an instantiated encryption rule, then*

1. *for each i, $\bar{x}_i \subseteq \mathcal{C} \cup \{Y\} \cup \bar{t}$.*

2. *$t\sigma$ has no more variables than t.*

Proof. There are two cases depending on whether the encryption rule $f(r, g(s, Y)) \rightarrow Y$ is of the symmetric-key or public-key form. Let u be the narrowing site, so that $t[u] = f(q, X)$, as in the definition of type V.

In the symmetric-key form, α and β in the rule are the same variable, so the step can be applied with $r = q$ if $\sigma = [X \rightarrow g(q, Y)]$. Any other substitution would be less general and would not be an AGU. As required for conclusion 1, any subterm of $X\sigma$ is either Y or occurs in $q < t$.

Conclusion 2 holds because σ deletes one variable, X, while it introduces one, Y.

In the public-key form, α and β in the rule are constants. Then σ maps q to α (so q is a variable or equal to α) and X to $g(\beta, Y)$. Hence the only subterms of terms in the range of σ are β and Y, giving us conclusion 1.

Conclusion 2 holds because σ is the composition of an AGU for q and α and the substitution $X \rightarrow g(\beta, Y)$ which deletes X as it introduces Y. ■

The non-expansion lemma is the most difficult part of the main result. It says that type V narrowing steps do not increase the number of distinct subexpressions in a term, except by adding constants. The proof is given in the Appendix.

Lemma 3.2 (Non-expansion) *If $t[\sigma, b \rightarrow Y]t'$ is type V then $|\bar{t'} \cup \mathcal{C}| \leq |\bar{t} \cup \mathcal{C}|$.*

Definition 3.4 (Renaming) *A substitution is called a renaming if it is one-to-one and every variable in its domain is mapped to another variable.*

Definition 3.5 (Distinct Terms) *Two terms s, t are distinct if there is no renaming σ such that $s\sigma = t$.*

Note that if there are two renamings σ and τ such that $s\sigma = t\tau$, then $\sigma' = \sigma\tau^{-1}$ is a renaming such that $s\sigma' = t$.

Lemma 3.3 (Boundedness) *If $\mathcal{S}$ is a set of distinct terms such that $|\bar{t}| \leq n$ for every $t \in \mathcal{S}, \mathcal{S}$ is finite.*

Proof. Recall that the sets of constants and operators are finite. Although the set of variables is infinite, we know that no term in $\mathcal{S}$ can have more than n variables in it. Hence, choose a set of variables of size n, say $\mathcal{W}$. For each term $t \in \mathcal{S}$, there is a renaming σ_t such that $t\sigma_t$ has only variables in $\mathcal{W}$. If we do this for each $t \in \mathcal{S}$, we map $\mathcal{S}$ into a set of terms that use only variables in $\mathcal{W}$, and which is no smaller than $\mathcal{S}$ because this mapping cannot merge distinct terms. We may assume that this renaming of variables has already occurred and $\mathcal{S}$ uses only variables in $\mathcal{W}$.

Let $\mathcal{N} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{W}$. Let B be the (finite) size of $\mathcal{N}$. We can view each term in $\mathcal{S}$ as a rooted directed tree, each of whose nodes is labelled by an element of $\mathcal{N}$.

The number of proper subterms of a term is one less than the number of nodes in the term-tree. Hence the number of nodes in each term-tree is at most $n + 1$. We now ask how many different term-trees there are having at most $n + 1$ nodes labelled from a set of size B. The answer is clearly finite, and it will serve as a loose upper bound for $|\mathcal{S}|$. ■

Now we can conclude that an infinite sequence of type V narrowing steps must enter a loop. In practice, after each type V step, one looks for opportunities to perform reductions, which also, obviously, cannot increase the number of subterms of a term, in the case of encryption rules. Hence we have:

Theorem 3.1 (Looping of Type V Sequences) *A sequence of type V narrowing steps, interspersed with reductions, can generate only a finite number of distinct terms.*

Proof. Immediate from the non-expansion and boundedness lemmas. ■

3.4 Narrowing Terminates for Encryption

Now let's put it all together. In order to solve an equation $s = t$, we must apply some sequence of substitutions and reductions to the equation (considered as a term with the operator '=') transforming it to an identity $r = r$.

Any sequence of substitutions and reductions based on encryption rules can be parsed into alternating phrases such that each phrase is

- a sequence of type C narrowing steps interspersed with some reductions, or
- a sequence of type V narrowing steps interspersed with some reductions.

The sequence of phrases may be preceded by a sequence of reductions and be followed at the end by a final substitution unifying the two sides of the equation. Any sequence of substitutions can be composed into a single substitution, and we assume this has been done to merge substitutions with other steps.

We have shown that a sequence of type C narrowing steps reduces the number of variables in a term, and is therefore bounded in length by the number of variables in the term. We have shown that a sequence of type V narrowing steps does not increase the number of variables in a term, and eventually enters a loop.

The only effect of a reduction based on an encryption rule is to delete part of the term structure, which does not increase the number of variables and cannot be performed ad infinitum. This means that a sequence of type C (or V) narrowing steps interspersed with reductions satisfies the same property as the type C (or V) steps alone. To summarize,

- type C phrases alternate with type V phrases,
- the number of variables in a term is reduced by each type C step and not increased by any other step,
- any type V sequence can be abbreviated to an equivalent finite non-looping sequence.

Since the total number of type C steps is bounded by the original number of variables in a term, and since type C and type V sequences alternate, any narrowing sequence that is loop-free must be finite. (Note that if the total number of type C steps were not bounded, the sequence could go back and forth forever between type C and type V phrases.)

A search algorithm can test all finite narrowing sequences because there are only a finite number of narrowing steps or reductions to choose from at a given time, so (by the König Lemma) the entire search tree is finite. We conclude that:

Theorem 3.2 (Termination Algorithm) *There exists a terminating algorithm to solve equations with respect to encryption rules.* ■

4 Conclusion

The termination result given here is based on a brute force approach to searching for solutions to an equation, namely trying all possible narrowing steps and performing reductions whenever possible. By checking for loops during sequences of type V narrowing steps, all search paths are bounded, so the search will eventually come to an end, with a solution if one is possible.

We have not given an *a priori* bound for the length of the search, and one based on these considerations would be too large to be practical. The value of the termination result is primarily that it shows that it is unnecessary to consider approaches that are incomplete, unless there is a significant efficiency gain by doing so, or the incompleteness is due to other operators.

This result is limited to abstract encryption rules expressing simple symmetric-key or public-key encryption. However, the technique could be extended to more rules, for these or other operators, if narrowing steps using those rules decrease some measure of a term which is not amplified by type C or V steps, and do not increase the number of variables in a term.

References

[1] J. W. Gray, III and J. McLean, "Using temporal logic to specify and verify cryptographic protocols (progress report)," *8th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995, 108-116.

[2] R. A. Kemmerer, "Analyzing encryption protocols using formal verification techniques," *IEEE J. Selected Areas in Communication*, Vol. 7, No. 4, May 1989.

[3] C. Meadows, "Using narrowing in the analysis of key management protocols," *1989 IEEE Symposium on Security and Privacy,* IEEE Computer Society, 1989, 138-147.

[4] J. K. Millen, "The Interrogator model," *1995 IEEE Symposium on Security and Privacy,* IEEE Computer Society Press, 1995, 251-160.

[5] A. W. Roscoe, "Modelling and verifying key-exchange protocols using CSP and FDR," *8th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995, 98-107.

[6] W. Snyder, *A Proof Theory for General Unification*, Birkhäuser, Boston, 1991.

Appendix: Non-expansion Lemma

Lemma 4.1 (Non-expansion) *If $t[\sigma, b \rightarrow Y]t'$ is of type V then $|\bar{t'} \cup \mathcal{C}| \leq |\bar{t} \cup \mathcal{C}|$.*

Proof: We will construct a function ϕ on $\bar{t} \cup \mathcal{C}$ whose range includes $\bar{t'} \cup \mathcal{C}$. Let

$$\phi(d) = d\sigma[b \rightarrow Y].$$

Since b is not a constant, $\phi(c) = c$ for any constant c, so $\mathcal{C} \subseteq \mathrm{rng}(\phi)$. It remains to show that $\bar{t'} \subseteq \mathrm{rng}(\phi)$.

Suppose $t'[u]$ is defined. We must show that $t'[u] \in \mathrm{rng}(\phi)$. Recall that $t'[u] = t\sigma[b \rightarrow Y][u]$. We also make an observation here that is used several times below: if $t\sigma[v] = b$ then v does not reach any X_i in t. For, if it did, $t\sigma[v] \leq x_i < b$.

Case 1. Suppose that u does not reach b in $t\sigma$ and that u does not reach any X_i in t. Using the address lemmas,

$$t'[u] = t\sigma[b \rightarrow Y][u] = t\sigma[u][b \rightarrow Y]$$
$$= t[u]\sigma[b \rightarrow Y] = \phi(t[u]).$$

Case 2. Suppose that u does not reach b in $t\sigma$ and that u does reach some X_i in t. As in Case 1,

$$t'[u] = t\sigma[b \rightarrow Y][u] = t\sigma[u][b \rightarrow Y].$$

Since u reaches X_i in t, $t\sigma[u] = x_i[w]$ where $u = vw$ and $t[v] = X_i$. Since $x_i < b$, we know that $b \not< x_i$, so $x_i[w][b \rightarrow Y] = x_i[w]$.

Now we must show that $x_i[w] \in \mathrm{rng}(\phi)$ for any w, that is, for x_i itself and any proper subterm of x_i. For x_i, we simply note that $\phi(X_i) = x_i$ and $X_i < t$.

Now let $s < x_i$. Using the Type V with Encryption lemma, $\bar{x_i} \subseteq \mathcal{C} \cup \{Y\} \cup \bar{t}$. If s is constant, $\phi(s) = s$ and we are done.

If $s = Y$, note that since $b < t\sigma$, there exists z with $t\sigma[z] = b$. Then

$$\phi(t[z]) = t[z]\sigma[b \rightarrow Y] = t\sigma[z][b \rightarrow Y] = Y = s.$$

Otherwise, $s < t$. Since $s < x_i < b$, $\phi(s) = s\sigma[b \rightarrow Y] = s$.

Case 3. Suppose that u reaches b in $t\sigma$.

In this case $t'[u] = t\sigma[b \rightarrow Y][u] = Y[w]$ where $u = vw$ and $t\sigma[v] = b$. But $Y[w]$ is undefined unless w is null. Hence $t\sigma[u] = b$ and we have

$$\phi(t[u]) = t[u]\sigma[b \rightarrow Y] = t\sigma[u][b \rightarrow Y]$$
$$= b[b \rightarrow Y] = Y = t'[u]. \blacksquare$$