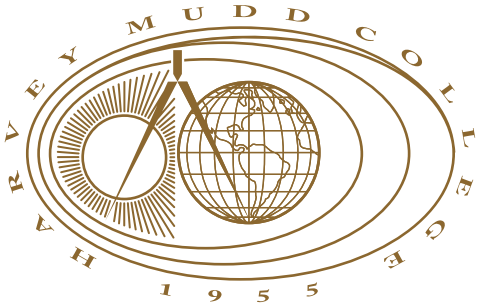
2001

# Structure Attacks in Cryptographic Protocols

Karl Mahlburg
*Harvey Mudd College*




# Structured Attacks on Cryptographic Protocols

by
Karl Mahlburg
Everett Bull, Advisor

Advisor: _____

Second Reader: _____

(Francis Su)

May 2001

Department of Mathematics

HARVEY MUDD
COLLEGE

**Abstract**

# Structured Attacks on Cryptographic Protocols

## by Karl Mahlburg

May 2001

Cryptographic protocols are in general difficult to analyze, and complicated attacks exposing security flaws have remained hidden years after a protocol is developed. Recently developed tools such as strand spaces and inductive logical proofs provide mechanical procedures for analyzing protocols. The key to these methods is that a generous upper bound on the activity of a malicious penetrator is often much easier to work with than a tighter bound.

However, these formalizations make strong assumptions about the algebraic structure of the cryptosystem that are never met in a real application. In this work, we show that an extended form of the strand space machinery can be used to analyze protocols which contain nontrivial algebraic structure, specifically that which arises from the XOR operation.

This work also serves as one of the first steps in reconciling computational and formal methods of analyzing cryptographic security.

# Table of Contents

## Acknowledgments

I would like to thank my advisor Rett Bull and my second reader Francis Su for their helpful discussions and patience.

# Chapter 1

## Introduction and Background

There are two primary branches in modern cryptography. One branch is concerned with encryption itself, encoding data in a secure way. The goal is to design cryptographic algorithms (or *cryptosystems*) that can be implemented efficiently but are difficult to reverse; or, in a complementary role, to search for weaknesses in existing methods. The other branch of study is concerned with using these algorithms to communicate securely. This is achieved with cryptographic *protocols*, in which parties exchange encrypted messages to ensure secure communication. The study of protocols includes the design of safe protocols, proofs that they function correctly, and the search for attacks on existing protocols.

These two branches have had little communication or collaboration, which has led to the misuse of encryption by both parties. A secure algorithm is useless when used in a flawed protocol, and a provably secure protocol may be vulnerable to simplistic attacks if a poor cryptosystem is implemented. This paper serves as one of the first steps to combine the two branches. We take an algebraic view of the structure of messages, and analyze protocols that use explicit encryption algorithms.

The discussion in this chapter is background material for the XOR protocols of chapter 2, and the strand space machinery is taken from [1]. See this work also for a more complete bibliography of sources for the study of protocols.

## *1.1 Cryptographic Protocols*

A security protocol is, loosely speaking, a sequence of messages between two or more parties (or *principals*), in which a publicly available encryption algorithm is used to ensure safe communication over a potentially insecure network. All adversarial activity on the network can be attributed to a single ubiquitous *penetrator*. The desired result of running a protocol is typically the authentication of identities or the distribution of new cryptographic keys. Each of the honest, or *regular* participants must follow a specific pattern of sent and received messages. Any deviations from the expected pattern of messages are interpreted as a security violation, and lead to the immediate termination of the protocol.

### *1.1.1 Message structure*

Before defining protocols more carefully, we need a clear view of the messages that may be used. Let $\mathsf{A}$ denote the set of all possible messages, or *terms*. This set represents all messages that can be constructed within a specified cryptosystem. The following algebraic properties are typically associated with $\mathsf{A}$:

- There is a set $\mathsf{T} \subset \mathsf{A}$ of texts, or atomic messages. $\mathsf{T}$ is the set of generators used that are used to construct all of the messages in $\mathsf{A}$. This set includes the names of all available principals, which are represented by $A_1, A_2, \ldots$, or by $A, B, \ldots$ when there are a small number of parties. The texts $\mathsf{T}$ also includes all *nonces*, or randomly chosen integers. A nonce is typically represented by $N_a$, where the subscript indicates that the nonce was generated by $A$; however, a nonce contains no identifying information.

- There is a set $\mathsf{K} \subset \mathsf{A}$ of cryptographic keys, which are disjoint from $\mathsf{T}$. Every key has an inverse key, which allows an encrypted message to be easily decrypted. The bijective key inverse operator $\mathsf{inv} : \mathsf{K} \to \mathsf{K}$ is typically written as

$\mathsf{inv}(K) = K^{-1}$. The inverse operator is idempotent, so that $(K^{-1})^{-1} = K$. If a key is its own inverse it is known as a *symmetric* key; in asymmetric cryptosystems keys exist in pairs $(K, K^{-1})$. A key consists of all data that are necessary to decrypt or encrypt a message for example. However, even a key that consists of multiple values is treated as a single unit.

- There is an encryption operation, $\mathsf{encr} : \mathsf{K} \times \mathsf{A} \rightarrow \mathsf{A}$ that is available to all participants. The operation is denoted by $\mathsf{encr}(K, g) = \{g\}_K$. There is a corresponding decryption operation that calculates $g$ from the inputs $\{g\}_K$ and $K^{-1}$.

- There is a concatenation operation, $\mathsf{join} : \mathsf{A} \times \mathsf{A} \rightarrow \mathsf{A}$ that is available to all participants. The operation is denoted by $\mathsf{join}(g, h) = gh$. A concatenation can be decomposed into its components by either of two separation operations: left-separation and right-separation. Therefore, either $g$ or $h$ can be obtained from $g\ h$.

- A is a free algebra with respect to encryption and concatenation. This means that if $g_1\ g_2 = h_1\ h_2$, then $g_1 = h_1$ and $g_2 = h_2$, and if $\{g_1\}_{K_1} = \{g_2\}_{K_2}$, then $g_1 = g_2$ and $K_1 = K_2$. The freeness of concatenation is violated in our intuitive notion of associativity in conjoined messages, but viewed formally, the order in which the terms were concatenated affects the resulting message. For example, $\mathsf{join}(\mathsf{join}(g, h), m)$ is different from $\mathsf{join}(g, \mathsf{join}(h, m))$. However, the parenthetical groupings of concatenated terms is unimportant in our context. Therefore, the groupings are ignored in our shorthand notation (both of the examples above are written as $g\ h\ m$).

A message that isn't of the form $g\ h$ is known as *simple*. This includes the atomic messages T, the keys K, and all messages $\{g\}_K$ that are encrypted at the outer level.

There is also a special subset of $\mathsf{A}_{\mathcal{P}} \subset \mathsf{A}$ associated with the penetrator. This set contains all the terms in $\mathsf{A}$ that the penetrator has knowledge of, including the names of regular participants, and any messages that are sent on the network. We denote the set of penetrator keys as $\mathsf{K}_{\mathcal{P}} = \mathsf{A}_{\mathcal{P}} \cap \mathsf{K}$ and the penetrator texts as $\mathsf{T}_{\mathcal{P}} = \mathsf{A}_{\mathcal{P}} \cap \mathsf{T}$. It is assumed that these are small proper subsets of $\mathsf{K}$ and $\mathsf{T}$, as we will need a supply of unguessable keys and nonces. The sets $\mathsf{K}_{\mathcal{P}}$ and $\mathsf{T}_{\mathcal{P}}$ also serve as a sort of generating set for $\mathsf{A}_{\mathcal{P}}$, for we assume that the penetrator's knowledge is closed under encryption and concatenation, as he can freely use these operations. However, the penetrator may also know messages that aren't generated in this way, such as encrypted messages emitted by regular participants.

When the cryptosystem uses public-private key pairs, we assume that there is a trusted key server that stores principal names and their associated long-term public keys. Any party on the network can contact the server and look up the public key of a registered principal. The penetrator can exploit this server by registering false names and the associated keys. He cannot usurp the identity of an honest principal through the server however. If the penetrator registers the name $C$, his actions under this guise are denoted by $P(C)$.

A subterm relation on $\mathsf{A}$ in the natural way: a term $g$ is a subterm of $g'$, denoted by $g \sqsubset g'$, if $g'$ can be constructed from $g$ through repeated concatenations and encryptions with other elements of $\mathsf{A}$. Specifically, for any terms $a, g, h \in \mathsf{A}$, and any key $K \in \mathsf{K}$, we have the following inductive definition:

1. $a \sqsubset a$

2. If $a \sqsubset g$, then $a \sqsubset g\, h$ and $a \sqsubset h\, g$

3. If $a \sqsubset g$, then $a \sqsubset \{g\}_K$

Note that encryption keys are not considered subterms of a message, so that $K' \sqsubset \{g\}_K$ only if $K' \sqsubset g$. In some models, the encryption key is treated as a subterm,

but in most cryptosystems, it is difficult to learn anything about the key from an encrypted message. We use the restricted definition because it will help us distinguish dangerous methods. Note that the order of concatenation is irrelevant when searching for an atomic subterm, such as a key or nonce. This is why we don't carefully track the structure of concatenations, as we are primarily concerned with messages that contain simple subterms. Examples of the subterm relation include: $g \sqsubset \{g\}_K$ and $\{g\}_K \sqsubset \{g\}_K \{\{h\}_K\}_{K'}$, but $K \not\sqsubset \{g\}_K$.

A is a well-founded partially ordered set under the subterm relation, since the relation is clearly transitive and any subset of terms $S \subset$ A has one or more minimal elements. For any term $g \in S$, we can apply at most a finite number of decryptions and separations to find its subterms, since $g$ was constructed by a finite number of concatenations and encryptions. Therefore, there cannot be infinite decreasing chains of messages.

We are now ready to formally define cryptographic protocols, and carefully state conditions for security.

### *1.1.2 Protocol Security*

A *protocol* is a finite ordered sequence of triples, $\{(A_i, B_i, g_i) | 1 \leq i \leq k\}$, where each element represents a message $g_i \in$ A from the principal $A_i$ to the principal $B_i$ at each step. As mentioned earlier, protocols are used to guarantee secure communication, but there are many possible definitions of security. The two common security goals are:

- **Secrecy**: Keys and nonces can be known only by the regular participants who use them, and are therefore trusted. In other words, the penetrator cannot learn decryption keys or randomly chosen values.

- **Authentication**: Every regular participant is communicating with whom they

intend. In other words, the penetrator cannot successfully masquerade as a regular participant or reroute communication in a harmful way.

These two security goals are often complementary: If secrecy is violated, then the penetrator may be able to use secret values to impersonate the regular participant that chose these values; if authentication is violated, then the penetrator may intercept and read secret data that was intended for an honest participant. In our proofs of security properties, secrecy usually implies authentication, for participants can use values that they know to be secret to deduce the identity of the party that could send a message with such values.

To illustrate the problems that arise in protocol security, consider a simple example: suppose that we would like to find a protocol that allows two principals to share a secure, random session key. A session key is a key shared between two principals that is only used for a limited time. Session keys are necessary because the long-term public-private key pairs are much more computationally expensive than many symmetric cryptosystems. Therefore, secure communication is usually established using the long-term keys, but the bulk of the messages are sent using session keys. These must be refreshed periodically, because the encryption is simple enough that brute-force guessing may be successful within several days. When the encryption key is constantly changing, and the penetrator is unaware of the changes, he cannot know which messages use the same key. The distribution of a session key is often accomplished by exchanging one or more nonces, for session keys can be easily generated using an algorithm that takes a random value as a seed.

A first attempt at such a protocol might consist of a single message sent from one principal to the other that contains the new key,

$$B \to A: \quad \{BK\}_{K_A}$$

After this protocol runs, both principals know $K$, since it was generated by $B$, and $A$ can read the encrypted message. Therefore, it might seem reasonable to use $K$ as a session key for communication between $A$ and $B$. However, the penetrator can easily impersonate an honest principal and send a similar message to $A$. All principal names are publicly available, as are the public encryption keys of the asymmetric key pairs, Thus, the penetrator uses the name $B$ and key $K_A$ to distribute a key of his choice, $K_p$, as follows

$$P \rightarrow A : \quad \{BK_p\}_{K_A}$$

Session keys usually contain no identifying information, so $A$ accepts any value and assumes that the name $B$ in the message was the sender. Since the penetrator can impersonate $B$, the protocol has no authentication guarantee.

Such a simple protocol fails because authentication requires messages that allow each principal to be confident in the identity of the other parties. Therefore, some of the messages must contain identifying information. The following example of a flawed protocol attempts to verify identities, but its failure illustrates some of the difficulties in designing and analyzing protocols.

The *Needham-Schroeder* protocol was designed to authenticate two principals by exchanging randomly chosen nonces in encrypted form [4].

$$\begin{aligned} A \rightarrow B : &\quad \{N_a\ A\}_{K_B} \\ B \rightarrow A : &\quad \{N_a\ N_b\}_{K_A} \\ A \rightarrow B : &\quad \{N_b\}_{K_B} \end{aligned}$$

Both $A$ and $B$ have a public-private key pair registered with the key server; $(K_A, K_A^{-1})$ and $(K_B, K_B^{-1})$ respectively. To construct the first message, the initiator $A$ generates a nonce $N_a$, looks up $B$'s public key $K_B$, and then encrypts the nonce and his own name. The responder $B$ decrypts this message using $K_B^{-1}$, selects a nonce $N_b$ of his own, and looks up the public key $K_A$ to construct the second message. $A$

decrypts this message with $K_A^{-1}$ and send out the final message in a similar way. For simplicity, we assume that nonces are always chosen uniquely, so that $N_a \neq N_b$ (this is a reasonable assumption, as the space of nonces is typically very large).

If the security goals of this protocol are met, then $N_a$ and $N_b$ will be secret values, while $A$ and $B$ each have an authentication guarantee for the other. Secrecy is easy to verify, since each message can only be read using the private key of an asymmetric key pair, and by assumption the penetrator cannot compromise the key server. Authentication requires more careful analysis. The initiator $A$ obtains the public key $K_B$ from the trusted key server, knowing that a message encrypted with $K_B$ can only be read by someone with knowledge of $K_B^{-1}$. This decryption key is known solely by $B$. Similarly, $B$ believes that a message encrypted with $K_A$ can only be read by $A$. Since $A$ sent out the nonce $N_a$ in an encrypted form, and received it back encrypted with another key, it may seem reasonable for $A$ to conclude that $B$ must have received and decrypted the first message to construct the second message, and similarly $B$ might conclude that $A$ read the second message to learn the value $N_b$ that he sent back in the third message. However, this conclusion is only partially correct, for the second message of the protocol contains no identifying information for $B$.

The Needham-Schroeder protocol in fact has an incomplete authentication guarantee for the initiator, since the penetrator can use messages sent by $A$ to communicate with another principal to make the following attack found by Gavin Lowe [2]:

$$
\begin{aligned}
A \rightarrow P(C): &\quad \{N_a\ A\}_{K_C} \\
P(C) \rightarrow B: &\quad \{N_a\ A\}_{K_B} \\
B \rightarrow A: &\quad \{N_a N_b\}_{K_A} \\
A \rightarrow P(C): &\quad \{N_b\}_{K_C} \\
P(C) \rightarrow B: &\quad \{N_b\}_{K_B}
\end{aligned}
$$

The penetrator has registered the false name $C$, and uses it to perform a "man-in-the-middle" attack. The messages sent and received by $A$ are $\{N_a\ A\}_{K_C}$ sent, $\{N_a\ N_b\}_{K_A}$ received, and $\{N_b\}_{K_C}$ sent. The nonce $N_b$ does not identify the principal who chose it, so $A$ assumes that it is also from $C$, and $A$ concludes that he has completed a run of the protocol with $C$. However, the messages received by $A$ were actually generated by $B$, which violates the expected authentication (we will define authentication more carefully later).

The penetrator cannot perform this attack in a slightly altered version, known as the *Needham-Schroeder-Lowe* protocol [3]:

$$\begin{aligned} A \rightarrow B: &\quad \{N_a\ A\}_{K_B} \\ B \rightarrow A: &\quad \{N_a\ N_b\ B\}_{K_A} \\ A \rightarrow B: &\quad \{N_b\}_{K_B} \end{aligned}$$

The only difference is the addition of the name in the second message. This additional information is enough to prevent the previous attack, where the penetrator uses a run of the protocol $A$ and $P(C)$ to establish communication between $A$ and $B$. The penetrator can again trick $B$ into sending the message $\{N_a N_b B\}_{K_A}$ to $A$, but since $A$ is expecting the name $C$, he immediately halts the protocol. However, this doesn't prove that the protocol has an authentication guarantee, for we have only seen how one possible attack fails. It is possible that there is another attack that does successfully compromise this protocol, and it is clearly not feasible to test all possible attacks. In the next section, we develop the additional machinery needed to prove the security of the protocol.

Lest the reader believe that the Needham-Schroeder protocol is just another trivial example, it bears noting that the protocol was introduced in 1977, but Gavin Lowe's attack was not published until 1995. Before that time, the protocol was used in many commercial software packages.

Another common attack is a "replay attack," where the penetrator remembers the messages that were sent in an honest run of a protocol and then establishes another run by sending the same messages. Such an attack can be prevented by using the fresh nonces seen in the Needham-Schroeder protocol, where each participant generates a new nonce for every run of the protocol and expects to get that same nonce back in encrypted form. If the penetrator tries to make a replay attack on the Needham-Schroeder protocol by resending the message $\{N_a\ A\}_{K_B}$ to $B$, the new run cannot be completed, for $B$ chooses a nonce $N_b'$ that is different from the nonces he had previously used. Thus the penetrator is unable to construct $\{N_b'\}_{K_B}$, and the protocol run can only be completed by contacting $A$. Then it becomes apparent to both parties that the penetrator is involved, for $A$ receives a message in the middle of a protocol run that he never initiated.

There is another type of replay attack that is available when a protocol has too many messages with a similar structure. For example, if two regular participants both transmit a nonce encrypted with a public key at some point in the protocol, the penetrator may be able to use such a message to impersonate an honest principal.

As the above examples illustrate, intuitive arguments about protocol security are often incorrect. The strand space machinery (and other similar methods) allow formal representation and analysis of protocol security.

### *1.2 Strand Spaces and Security Properties*

The communication in cryptographic protocols lends itself to a graphlike representation with directed edges that is ased on causal relationships. Since each step of a protocol must occur in order, and each received message must have a sender, we can follow these relationships to track messages. Specifically, we are interested in finding the origination of messages that contain secret values, for it is the penetrator's knowledge of such values that violates security.

### *1.2.1 Strand Spaces*

With each principal in a protocol we associate a *strand*, which is a directed path containing one node for each step of the protocol involving that principal. Each node is labeled by a signed term, where the term is the message at the step of the protocol that corresponds to the node. The terms are given a $+$ sign for transmitted messages and a $-$ sign for received messages. We denote the label of node $n$ as term$(n)$. The edges are denoted by $n' \Rightarrow n$ when $n'$ is the unique node that immediately precedes $n$ on a strand. If $n$ follows $\bar{n}$ at some point on the strand (not necessarily immediate precedence), then $\bar{n} \Rightarrow^+ n$.

The *trace* of a strand is just the sequence of signed terms seen by the principal. For a given protocol, the regular strands correspond to the actions of regular participants. In the Needham-Schroeder protocol, there are two types of regular strands:

1. "Initiator strands" $s \in \mathrm{Init}\,[A, B, N_a, N_b]$ with trace:

$$\langle +\{N_a\ A\}_{K_B}, -\{N_a\ N_b\}_{K_A}, +\{N_b\}_{K_B} \rangle$$

   where $A, B$ are names in $\mathsf{T}$, and $N_a, N_b$ are nonces. Since principal names are mapped injectively into public-private key pairs by the key server, $K_A$ and $K_B$ are uniquely determined by the choice of $A$ and $B$ and are therefore not included as parameters. The principal $A$ is associated with these strands, and is known as the "initiator."

2. "Responder strands" $s \in \mathrm{Resp}\,[A, B, N_a, N_b]$ with trace:

$$\langle -\{N_a\ A\}_{K_B}, +\{N_a\ N_b\}_{K_A}, -\{N_b\}_{K_B} \rangle$$

   As before, $A, B \in \mathsf{T}$, and $N_a, N_b$ are nonces. The principal $B$ is associated with these strands, and is known as the "responder."

In a specific run of the protocol, the names and values of the nonces on the initiator and responder strand correspond; however, an arbitrary collection of strands need not form a valid protocol run.

When we wish to consider the collection of strands with all possible values in one of the parameters, we write $*$ in place of a specified value. For example, Init $[A, *, *, N]$ is the set of initiator strands in which the initiator $A$ receives the nonce $N$, with all possible principals as the responder, and all possible nonces chosen by $A$. When all the parameters are free, we shorten the notation to just Init or Resp. The collection of all possible initiator and responder strands is called a *NS strand space*, and is denoted by $\Sigma_{NS}$; in general, the collection of all strands for an unspecified protocol is just a *strand space* $\Sigma$.

It will be useful to have concepts of message origination shortly:

- A term $g$ *originates* on a node $n$ if $g \sqsubset \text{term}(n)$ and $g \not\sqsubset \text{term}(n')$ for any node $n'$ such that $n' \Rightarrow^+ n$. Thus $n$ is the first node of some strand that such that $n$ contains $g$ as a subterm.

- For a set of messages $M \subset \mathsf{A}$, a node $n$ is an *entry point* to $M$ if $\text{term}(n) \in M$ but $\text{term}(n') \notin M$ for any $n'$ such that $n' \Rightarrow^+ n$. Thus $n$ is the first node of some strand such that $n$ contains any $g \in M$ as a subterm.

To help us model protocol runs, we also need to represent communication between strands. Therefore we add an additional type of connecting edge between nodes of different strands, representing communication over the network. Specifically, if $n$ is a node with a negative signed term, then there must be some unique node on another strand that sent the message, i.e. there is a unique $n'$ with the same term and positive sign, where $n' \not\Rightarrow^+ n$. Then there is an edge $n' \rightarrow n$.

The directed graph of $\Rightarrow$ and $\rightarrow$ edges has the property that each node is at the end of at most one edge of both types, since there is a unique causal precursor and a

unique sender if the node received a message. Each node is also the beginning of at most one $\Rightarrow$ edge, since the protocols run deterministically; however, a node may be the beginning of any number of $\rightarrow$ edges, for there may be multiple recipients of a message.

The actions of the penetrator are modelled through a collection $\mathcal{P}$ of *penetrator strands*. These penetrator strands apply to every protocol, since the penetrator actions are assumed to be universal. The strands in $\mathcal{P}$ have one of the following traces:

- **M**. *Text message*: $\langle +t \rangle$ *where* $t \in \mathsf{T}_{\mathcal{P}}$.
- **C**. *Concatenation*: $\langle -g,\ -h,\ +g\ h \rangle$.
- **S**. *Separation*: $\langle -g\ h,\ +g,\ +h \rangle$.
- **K**. *Key*: $\langle +K \rangle$ *where* $K \in \mathsf{K}_{\mathcal{P}}$.
- **E**. *Encryption*: $\langle -K,\ -g,\ +\{g\}_K \rangle$.
- **D**. *Decryption*: $\langle -K^{-1},\ -\{g\}_K,\ +g \rangle$.

A *penetrator node* is a node on some penetrator strand. Unlike regular participants, who have exactly one strand associated with each run of the protocol, all of the penetrator strands are associated with the omnipresent penetrator. An *infiltrated strand space* is the union of a strand space $\Sigma$ and the penetrator strands $\mathcal{P}$.

The definition of the key and text strands illuminates the use of the sets $\mathsf{T}_{\mathcal{P}}$ and $\mathsf{K}_{\mathcal{P}}$. These penetrator strands represent random guesses of keys and nonces made by the penetrator, and also the public values that he can access. Therefore, $\mathsf{T}_{\mathcal{P}}$ contains some collection of principal names and all of the nonce values that the penetrator guesses during the run of the protocol, and $\mathsf{K}_{\mathcal{P}}$ contains any collection of public keys and all of the random keys that the penetrator guesss. Any keys and

nonces (secret or otherwise) that the penetrator can only learn from the protocol are not included in the penetrator set. This restriction is made because we do not have a precise concept of time in the strand model, and causality is a problem if the penetrator has prior knowledge of a secret key before any messages have even been transmitted. Instead, we require the penetrator to derive any subsequent use of a value learned from a regular participant from the original message. Then all penetrator activity involving this value can be traced back to the origination point on a regular node, which reflects causality.

Now we can define a useful set of logically connected nodes. A *bundle* is a finite set of nodes and edges $\mathcal{C}$ such that:

- For every node $n \in \mathcal{C}$, the precedence $\bar{n} \Rightarrow^+ n$ implies that $\bar{n} \in \mathcal{C}$ as well.
- For every node $n \in \mathcal{C}$ with term$(n) = -g$ there is a unique corresponding node $n' \in \mathcal{C}$ (on another strand) such that term$(n) = +g$.

In words, if some node is in the bundle, then all of the preceding steps on that node's strand must also be included; and every time a message is received there must have been a sender. However, a node that transmits a message in a bundle isn't required to find a recipient, and the communication may just disappear into the network.

A bundle contains only the penetrator nodes that directly affect the communication between regular participants. The effects of the penetrator's interference may be trivial, so that even for a flawed protocol there exist bundles in which the regular participants behave correctly. For example, the penetrator may intercept a message $g$, apply a **E**-strand with some key $K$ to get $\{g\}_K$, and then immediately run a **D**-strand to send the original message $g$ to its intended recipient. In fact, in most bundles the penetrator's actions are ineffective. However, if there is an attack possible on the protocol, then for each set of parameter values on the regu-

lar strands, there is at least one bundle in the infiltrated strand space in which the penetrator successfully performs the attack.

Bundles are useful because formal security properties can be easily stated in terms of the strands contained in some bundle. For example, a protocol preserves the secrecy of some set of terms $S \subset \mathsf{A}$ if and only if there is no bundle in the infiltrated strand space $\Sigma, \mathcal{P}$ such that a penetrator node contains a term $g \in S$ in unencrypted form. Conversely, if the protocol is flawed and there exists an attack in which the penetrator learns some secret value $g \in S$, then there must be some regular node that transmits a term $h$ with $g \sqsubset h$ such that the penetrator can deconstruct $h$ to obtain $g$. Then there is a bundle in which a penetrator node has term $g$, for the penetrator need only intercept $h$ from the network and send it through the appropriate sequence of **D** and **S**-strands.

Authentication guarantees can also be formalized with bundles. In a secure protocol each regular participant should be able to conclude the existence of the other principals. As with secrecy, the authentication guarantees are stated in terms of strands in a bundle. For example, an authentication guarantee for the initiator in the Needham-Schroeder protocol can be stated as follows: for any bundle $\mathcal{C} \in \Sigma_{NS}, \mathcal{P}$ that contains an initiator strand $s \in \mathsf{Init}\,[A, B, N_a, N_b]$ there is also exactly one responder strand $s' \in \mathsf{Resp}\,[A, B, N_b]$.

We need one more piece of machinery before proving security properties in the strand space context.

### *1.2.2 Honest Ideals*

The security properties of protocols describe the honest actions that function correctly regardless of penetrator activity. However, it is not possible to consider all possible sequences of penetrator actions, for an arbitrary finite number of strands may be involved. Even with small bounds on the extent of penetrator actions, the

number of random keys and nonces create a prohibitively large set of activities. Instead, we use the algebraic structure of messages to place an upper bound on all finite sequences of penetrator actions. Fortunately, the security properties often hold even when the penetrator is at the limit of these bounds.

First, define an *ideal* $I = I_{\mathsf{k}}[S]$, constructed from a subset of keys $\mathsf{k} \subset \mathsf{K}$ and a set of secret terms $S \subset \mathsf{T} \cup \mathsf{A}$. The name is no accident; the structure is similar to an algebraic ring ideal. The ideal $I$ is defined to be the smallest set containing $S$ that is constructed inductively in the following way: If $g \in I, h \in \mathsf{A}$ and $K \in \mathsf{k}$, then

- $h\ g \in I$ and $g\ h \in I$
- $\{h\}_K \in I$.

Now let $S$ be a set of simple terms such that $S \cap \mathsf{A}_{\mathcal{P}} = \emptyset$, and if $g \in S$ has the form $\{h\}_K$, then $K \in S$. The secret set usually contains just keys and nonces, which is of this form. Define the key set $\mathsf{k}^{-1} = \mathsf{K} \setminus S$. This final condition ensures that if the private key $K^{-1}$ of a public-private key pair $(K, K^{-1})$ is secret, then $K^{-1} \in \mathsf{k}$. If there is a symmetric key $K_{AB} \in S$, the condition gives $K_{AB} \notin \mathsf{k}$. This definition of $\mathsf{k}$ is used because a carefully chosen ideal is meant to bound the set of messages that are potentially dangerous. The ideal is constructed from the secret set, and loosely speaking, if a message can be deconstructed to some value in $S$, then it's dangerous. Therefore, when the private key $K^{-1}$ is secret and thus $K$ is public, the penetrator could use $K$ to decrypt a message of the form $\{g\}_{K^{-1}}$. If $g$ contains information about $S$, then this a dangerous message, so we include such messages in the ideal. However, a symmetric key can only be decrypted by itself, so if $K_{AB}$ is secret, we need not worry about messages encrypted with this key.

We have now reached the most important result of our analysis, which uses the ideal construction to prove secrecy:

**Claim.** *Let $S$ be the secret set, $\mathsf{k}^{-1} = \mathsf{K} \setminus S$, and $I = I_{\mathsf{k}}[S]$. Then the complement $I^{\mathsf{c}} = \mathsf{A} \setminus I$*

*is closed under penetrator actions and is disjoint from $S$.*

*Proof.* We will show that any penetrator node $p$ that is an entry point to $I$ must have negative sign, i.e. term$(p) = -g$. This means that the penetrator can only construct a message in $I$ after receiving some other message in $I$. The contrapositive of this result gives the closure of $I^{\mathsf{c}}$ on penetrator strands.

Suppose to the contrary that $p$ is a penetrator entry node to $I$ with positive term. This is not possible on any of the penetrator strands:

- **M**. The trace $\langle +t \rangle$ has an entry point to $I$ only if $t \in S \cap \mathsf{T}$. But then $t \notin \mathsf{T}_{\mathcal{P}}$ as required by the definition of the text strand, since $S \cap \mathsf{T}_{\mathcal{P}} = \emptyset$.

- **C**. This has trace $\langle -g,\ -h,\ +g\ h \rangle$, so we must have the positive term $gh \in I$ but $g \notin I$ and $h \notin I$. But the set $S$ contains only simple terms, so $g\ h \notin S$, and therefore one of $g$ or $h$ must be in $S \subset I$, so one of the negative terms is the entry point instead.

- **S**. One of $g$ or $h$ must be in $I$, since they are the positive terms in the trace $\langle -g\ h,\ +g,\ +h \rangle$. However, by the definition of an ideal, the concatenated term is also in $I$, so it is the entry point.

- **K**. The trace $\langle +K \rangle$ has an entry point to $I$ only if $K \in S \cap \mathsf{K}$. But then $K \notin \mathsf{K}_{\mathcal{P}}$, violating the definition of the key strand.

- **E**. For the trace $\langle -K,\ -g,\ +\{g\}_K \rangle$, we must have the encrypted term $\{g\}_K \in I$ and $g, K \notin I$. However, if $K \notin S$, then $\{g\}_K \notin S$, so $g \in I$, and otherwise $K \in S \subset I$. Therefore one of the negative terms is an entry point to $I$.

- **D**. With trace $\langle -K^{-1},\ -\{g\}_K,\ +g \rangle$, we must have $g \in I$. If $K^{-1} \notin S$, then $K \in \mathsf{k}$ and hence $\{g\}_K \in I$; otherwise $K^{-1} \in S \subset I$, so one of the negative terms is the entry point.

□

This result is used to prove the security for a protocol that has no regular entry point to the set $I$, as long as the penetrator has no prior knowledge, so $I \cap \mathsf{A}_{\mathcal{P}} = \emptyset$. Then the $S$ remains secret because all of the terms available to the penetrator are in $I^{\mathsf{c}}$, which is closed under penetrator actions. Therefore he is unable to construct any term $g \in I$, and specifically no $g \in S \subset I$. Thus, the protocol preserves the secrecy of $S$.

Authentication guarantees are proved using the secrecy of certain values and the structure of regular messages. Any message that is encrypted with a secret key was generated a regular participant who knew that key. Recall that to prove authentication, we assume the existence of one type of strand and deduce the existence of another corresponding strand. This is accomplished by using the graph structure and tracing back the received messages. A demonstration of this secrecy and authentication analysis on the Needham-Schroeder protocol follows.

### *1.2.3 Needham-Schroeder-Lowe security*

Recall the protocol

$$\begin{aligned} A \to B: &\quad \{N_a\ A\}_{K_B} \\ B \to A: &\quad \{N_a\ N_b\ B\}_{K_A} \\ A \to B: &\quad \{N_b\}_{K_B} \end{aligned}$$

The goal of the protocol is to validate identities and share the nonces $N_a$ and $N_b$. The secret set is thus $S = \{N_a, N_b, K_A^{-1}, K_B^{-1}\}$, and the key set $\mathsf{k}$ is defined as $\mathsf{k} = \mathsf{K} \setminus \{K_A, K_B\}$. Consider the ideal $I = I_{\mathsf{k}}[S]$, and assume that $\mathsf{A}_{\mathcal{P}} \subset I^{\mathsf{c}}$ (equivalently, assume that $\mathsf{K}_{\mathcal{P}} \cap S = \mathsf{T}_{\mathcal{P}} \cap S = \emptyset$). In every bundle $\mathcal{C} \in \Sigma_{NSL,\mathcal{P}}$, each regular node $n$ has term$(n) \in I^{\mathsf{c}}$. The penetrator's specific actions are irrelevant, since $I^{\mathsf{c}}$ is closed under his available strands. Therefore, in every bundle $\mathcal{C}$, each penetrator

node $p \in \mathcal{C}$ has term$(p) \in I^{\mathsf{c}}$, so term$(p) \notin S \subset I$. Hence, the protocol preserves the secrecy of $S$.

Now we prove the initiator's authentication guarantee. This guarantee is stated as follows: In any bundle $\mathcal{C} \in \Sigma_{NSL}, \mathcal{P}$, if there exists a strand $s \in \text{Init}\,[A, B, N_a, N_b]$, then there exists a strand $r \in \text{Resp}\,[A, B, N_a, N_b]$ such that at least the first two nodes of $r$ are in $\mathcal{C}$. We know nothing about the final node of $r$, since the bundle is still valid even if the initiator's final message $\{N_b\}_{K_B}$ disappears into the network.

To prove this guarantee, we first use the closure of $I^{\mathsf{c}}$ and claim that there is no message $g \in I$ that originates on a penetrator node. By the closure of $I^{\mathsf{c}}$, any such term must originate negatively since the penetrator has no initial knowledge of $I$. But no terms in $I$ are emitted by a regular node in the protocol, so there are no entry points to $I$ anywhere in the bundle. Thus, all of the penetrator nodes are in $I^{\mathsf{c}}$. Now consider the message $\{N_a\ N_b\ B\}_{K_A}$, which is received by $A$ after he sends the first message of the protocol. If the message had been constructed by a penetrator, then he would have a node containing $K_A \in I$, but this contradicts the argument just made.

Therefore, when the initiator receives the message $\{N_a\ N_b\ B\}_{K_A}$, he knows that it must have originated on a regular node. Furthermore, it must have been the second node of a responder strand $r$, since no other regular nodes generate messages of this form. The values and names used in the message imply that $r \in \text{Resp}\,[A, B, N_a, N_b]$, which is the guarantee we desired.

Recall that the original Needham-Schroeder protocol lacked such a guarantee. We argued before that the deficiency is because the second message of the protocol is $\{N_a\ N_b\}_{K_A}$, which contains no identifying information. We can now see more precisely why this protocol is flawed. The arguments above show that this message must have been sent by a responder strand $r$, but the initiator can only conclude that $r \in \text{Resp}\,[A, X, N_a, N_b]$ for some principal $X$. This deficiency is reflected in the attack seen earlier, since the penetrator was able to establish a different responder

strand than the initiator expected.

Slightly more sophisticated arguments show that the responder has a full authentication guarantee in both the NS and NSL protocols. It is not especially illuminating to present them in detail. Briefly, the responder receives two messages, and he must show that they both originated on the same initiator strand. He knows that since his nonce $N_b$ was sent in encrypted form, it only could have been read by $A$. But $B$ also knows that the nonce $N_a$ was read by $A$ and matched the nonce sent in the first message (since otherwise the protocol would have halted and the third message wouldn't have been sent). Since nonces are uniquely originating, the initiator strand on which $N_a$ was initially chosen is the same strand as the one that received $N_a$ and $N_b$ in encrypted form, which proves the authentication result.

The above arguments also illustrate why a protocol is more likely to be flawed when it has several messages of the same form. Authentication guarantees are more difficult to prove in this case, as there may be many possible origination points for some messages. This makes it more difficult to deduce that the other strands in the bundle have the expected form.

This strand space machinery works well on *unstructured* protocols, where the message space is a free algebra. However, it isn't clear how effective it will be in *structured protocols*, where the algebra isn't free. In our original work of the next chapter, we develop extensions to the ideal structure that allow us to prove the security properties of an important class of structured protocols.

# Chapter 2

## XOR

The XOR operation, $\oplus$, is defined on two bit strings of the same length by performing the binary XOR operation pairwise on the bits of each string (which is a function on two bits that evaluates as *true* when exactly one of the inputs is *true*, and evaluates *false* otherwise). Thus, for bit strings $m = b_1 b_2 \dots b_k$ and $n = c_1 c_2 \dots c_k$ where $b_i, c_i \in \{0, 1\}$, XOR is defined by $m \oplus n = (b_1 \oplus c_1)(b_2 \oplus c_2) \dots (b_k \oplus c_k)$. This operation inherits commutativity and associativity from the binary XOR operation, since the bit strings use what is essentially an exterior product of the bit structure. Additionally, XOR has the identity $(m \oplus n) \oplus n = m$ for any bit strings $m, n$.

There are several cryptosystems that use XOR by transmitting a message $m$ as $m \oplus K$ for some secret $K$. The safest methods use *one-time encryption*, where a new key is used for each encryption. If the key is chosen randomly each time, then the encryption is provably unbreakable. However, the recipient must know every encryption key to obtain the original message, and distributing a large number of keys is potentially dangerous. XOR can also be used effectively in an iterated encryption system, where the first message $m_1$ is encrypted as $m_1 \oplus K$, and each successive message is combined with the previous message, so that $m_2$ is encrypted as $m_2 \oplus m_1$, $m_3$ as $m_3 \oplus m_2$, and so on. Unraveling this sequence to get the original messages requires the receipt of all of the encrypted terms and the knowledge of $K$. It is usually necessary to use varied keys when XOR is used for encryption, for if the same key were used repeatedly to encrypt many different messages, then a malicious penetrator may be able to determine portions of the messages or even the encryption key itself. For example, if $m, n$ both contain similar file-type identifiers,

then the penetrator could compute $(m \oplus K) \oplus (n \oplus K) = m \oplus n$. This result will contain long strings of 0-bits where $m$ and $n$ share identifying codes, which the penetrator can use to guess the filetype. If his guess is correct, then he can use that file-type identifier and analyze $m \oplus K$ to deduce portions of $K$.

The structure of XOR cannot easily be analyzed in the current strand space model. To illustrate the issues that arise, consider the following protocol that allows $A$ to distribute a secret value $N_a$ without relying on a key server:

$$\begin{aligned} A \rightarrow B: &\quad \{N_a\}_{K_A} \\ B \rightarrow A: &\quad \{\{N_a\}_{K_A}\}_{K_B} \\ A \rightarrow B: &\quad \{N_a\}_{K_B} \end{aligned}$$

This protocol is similar to ones that are used in public key exchange software. Note that this protocol doesn't operate in a free algebra of messages, as the encryption is assumed to be commutative; i.e., the interior encryption can be removed without affecting the exterior. Regardless of whether the encryption is symmetric or asymmetric, this protocol is known to be vulnerable to an attack where the penetrator receives $\{N_a\}_{K_A}$ and just sends it back to $A$ without adding another layer of encryption. Then $A$ is tricked into sending out $N_a$ unencrypted in the final step. This attack is prevented if $A$ can distinguish the layers of encryption and realize that the message he received has the wrong form, but since the algebra is no longer free, this may not be possible. Fortunately, this attack can also be prevented if $A$ just checks that he received a different message than the one he sent out. However, if XOR is used for encryption, then this protocol has a deeper exploitable structure. The sequence of protocol messages with XOR encryption is

$$\begin{aligned} A \rightarrow B: &\quad N_a \oplus K_A \\ B \rightarrow A: &\quad N_a \oplus K_A \oplus K_B \\ A \rightarrow B: &\quad N_a \oplus K_B \end{aligned}$$

The penetrator can calculate $(N_a \oplus K_A) \oplus (N_a \oplus K_A \oplus K_B) \oplus (N_a \oplus K_B) = N_a$, violating the secrecy of the value $N_a$. This attack is not possible if the encryption is merely commutative, as the structure of XOR enables the penetrator to create new keys that are closely related to old keys.

The strand space machinery has proven to be a powerful method of analyzing protocols, and we do not wish to abandon it, but it must be extended to include the structure of XOR. Our eventual goal is to use modified forms of the ideal construction to analyze protocols that use XOR.

### *2.1 XOR Exchange Protocols*

An *XOR exchange protocol* is a protocol in which the symmetric cryptosystem is assumed to be unbreakable, and the only nontrivial identities on the message algebra are due to the use of XOR on a subset of keys. An example of such a protocol is an altered version of the *TMN protocol* [5], in which a trusted server uses XOR to distribute a two-party session key

$$
\begin{aligned}
A \to S: &\quad \{m_1\ B\ K_a\}_{K_{AS}} \\
S \to B: &\quad A\text{req} \\
B \to S: &\quad \{m_3\ A\ K_b\}_{K_{BS}} \\
S \to B: &\quad K_a \oplus K_b \\
A \to B: &\quad \{A\}_{K_b} \\
B \to A: &\quad \{A\ B\}_{K_b}
\end{aligned}
$$

The messages $m_1, m_3$ are just identifiers that prevent the penetrator from switching the order of the messages in a replay attack, and $K_{AS}$, $K_{BS}$ are symmetric keys shared between the server and a regular participant. The goal of the protocol is to establish a session key $K_b$. The reader may question why XOR is used at all, for the protocol would be provably secure if the XOR term were replaced by $\{K_a\}_{K_{AS}}$.

The reason is that XOR can be calculated very efficiently, and if the protocol can be proven secure with XOR, then computation is saved.

Although this may seem like a limited use of XOR, such protocols still have security concerns beyond those covered by the current ideal theory. To demonstrate the problem, consider a protocol with a secret symmetric key $K$, and a symmetric key $K'$ known by the penetrator. Further, suppose that the key $K \oplus K'$ is not a secret key. If we construct an ideal $I$ around the secret set, then $K \in I$, and $K', K \oplus K' \notin I$. However, if a regular participant emits the key $K \oplus K' \in I^{\mathsf{c}}$, the penetrator can calculate $K = (K \oplus K') \oplus K'$. This means that the current definition of ideal no longer satisfies the closure properties that allowed us to prove security, and the structure of XOR needs to be included.

We would like to prove that if $K_a$ and $K_b$ are secret from the penetrator in the TMN protocol, then the secrecy is maintained even when $K_a \oplus K_b$ is transmitted. We need a better understanding of the XOR structure in order to formalize this notion.

### *2.1.1 XOR Group Structure*

Suppose there is a subset of keys $\mathsf{X} \subset \mathsf{K}$ on which the XOR operation is defined and closed. Thus $K_1, K_2 \in \mathsf{X}$ implies that $K_1 \oplus K_2 \in \mathsf{X}$. All of the keys in this set must have the same bit-length $l$, and we assume that the converse is also true, so that all bit strings of this length are necessarily in $\mathsf{X}$. This assumption is made to avoid type ambiguities, and since keys are involved in encryption and other message types are not, we consider the most interesting structure. Generalizations are discussed in section 2.1.6, where XOR is allowed to act on texts as well. The XOR product of two sets of keys is defined in the natural way: if $A, B \subset \mathsf{X}$, then $A \oplus B = \{a \oplus b \mid a \in A, b \in B\}$.

Since XOR preserves bit length, $\mathsf{X}$ is closed and has a group structure under the

XOR operation. Note that every element is nilpotent with degree two, and that the group operation is commutative. Therefore, the XOR structure is isomorphic to the abelian group $(\mathbb{Z}_2)^l$, with the natural isomorphism that maps each bit from the bit string to the corresponding element in the the product. This space is also isomorphic to a vector space of dimension $l$ over the field of two elements. We allude to the group structure and call the result of combining the two keys under XOR the *product* of the keys.

Suppose $S_{\mathsf{X}} \subset \mathsf{X}$ is a subset of $n$ keys, with $S_{\mathsf{X}} = \{K_1, K_2, \ldots, K_n\}$. Since $\mathsf{X}$ is a vector space, the *span* of $S_{\mathsf{X}}$ is a vector subspace given by:

$$\langle S_{\mathsf{X}} \rangle = \left\{ K \in \mathsf{X} \;\middle|\; K = \bigoplus_{k=1}^{m} K_{i_k}, 1 \leq m \leq n, i_k \in [1, \ldots, n] \right\}$$

This is the set of all product keys that can be calculated from $S_{\mathsf{X}}$. Note that we don't require the indices $i_k$ to be distinct. In fact, the trivial or identity key (all 0-bits) is contained in the span as the product $K_1 \oplus K_1$, and is a vital component in the group structure. However, the identity key is implicitly excluded whenever we discuss encryption keys, and it is never chosen as a random key. The identity key is denoted by $K_{\mathsf{id}}$. Every nontrivial key in $\langle S_{\mathsf{X}} \rangle$ can be represented as a product with no repeated keys, since any repeated keys cancel (e.g., $K_1 \oplus K_2 \oplus K_1 = K_2$). As a subspace, $\langle S_{\mathsf{X}} \rangle$ is closed, which can also be seen by considering the product of any two elements. Such a product is again a product of some sequence of keys in $S_{\mathsf{X}}$, and we are again left with either zero or one copies of each $K_i$ in the final product.

The span of a more general subset $H \subset \mathsf{A}$ can also be defined as $\langle H \rangle = H \cup \langle H \cap \mathsf{X} \rangle$. The span of the XOR keys is included, and the terms without XOR structure are present without adding any additional terms.

The set $S_{\mathsf{X}}$ is *independent* if no key in $S_{\mathsf{X}}$ can be written as a product of other keys in $S_{\mathsf{X}}$. Since the span of a set of keys contains all possible products, the condition of independence for $S_{\mathsf{X}}$ is equivalent to $K \notin \langle S_{\mathsf{X}} - \{K\} \rangle$ for all $K \in S_{\mathsf{X}}$. It is easier

to describe $\langle S_{\mathsf{X}} \rangle$ when $S_{\mathsf{X}}$ is independent, for then each distinct product in $\langle S_{\mathsf{X}} \rangle$ is distinct. $S_{\mathsf{X}}$ need not be independent, and there may be intentional relations between terms of $S_{\mathsf{X}}$. However, if each of the $n$ keys in $S_{\mathsf{X}}$ are chosen randomly, and $n$ is small compared to $l$, then the keys are likely to be independent. This is because after $K_1$ is chosen freely, we must choose $K_2 \in \mathsf{X} \setminus \langle \{K_1\} \rangle$, then choose $K_3 \in \mathsf{X} \setminus \langle \{K_1, K_2\} \rangle$, and so on. If $S_{\mathsf{X}}$ isn't independent, then there is some basis $B \subset S_{\mathsf{X}}$ such that $\langle S_{\mathsf{X}} \rangle = \langle B \rangle$, for a vector subspace always has an independent basis. If $n > l$, then $S_{\mathsf{X}}$ is dependent.

We make the following important assumption: a key that is randomly chosen by a principal is independent from all other keys being used by active principals in the protocol. This assumption requires that the dimension of the keyspace be much larger than the number of keys used in the protocol, which means that the logarithm of the size of $\mathsf{X}$ must be large relative to the number of keys used in the protocol. Fortunately, a typical protocol uses only a handful of keys, so the keyspace can be of reasonable size. This is a stronger version of the assumption of uniquely originating nonces in previous work. This assumption serves the same purpose; namely, that there aren't any unexpected relations between the values chosen by regular participants.

### *2.1.2 XOR Strand*

The XOR operation is available to all parties, so that any principal can calculate $K \oplus K'$ if he knows $K, K' \in \mathsf{X}$. We define the XOR strand in a way that restricts the penetrator strongly, and may seem counter-intuitive.

$\mathbf{X}$. *XOR*: $\langle -K, -K', +K \oplus K' \rangle$ *where* $K, K' \in \mathsf{X} \setminus \mathsf{K}_{\mathcal{P}}$.

The necessity of this definition will be clearer after a discussion of the use of the strand in modeling penetrator actions.

First, notice that the algebra of terms with XOR is no longer free, and therefore has more complicated structure than encryptions and concatenations. Recall that before, the subterm relation was defined by $a \sqsubset b$ if $b$ can be constructed from $a$ through some sequence of concatenations or encryptions. However, if this definition is extended to XOR, then $K$ would be a subterm of $(K \oplus K') \oplus K'$ for any $K'$, and thus each key would be a subterm of every other key. This would collapse the XOR keys and diminish the usefulness of the subterm relation in tracing messages. Instead, we define $K \sqsubset K \oplus K'$ if and only if $K' = K_{\mathsf{id}}$. The product $K \oplus K'$ is just another key, which is distinct from $K$.

As before, we have introduced the $\mathbf{X}$-strand to model the penetrator's actions. Our goal is to bound the penetrator activitiy by constructing an ideal $I$ around a secret set of messages, and we would like to prove that the complement of the ideal $I^{\mathsf{c}}$ is closed under penetrator actions. If the $\mathbf{X}$-strand were defined so that the penetrator had free use of XOR, then we would be making the incorrect assumption that the penetrator can efficiently guess the value of any random, secret key. This is because the penetrator is free to make random guesses of secret keys, and he could easily guess a set of keys that form a basis for all bit strings of length $l$, such as $\mathsf{K}_{\mathcal{P}} = \{1000\ldots, 0100\ldots, 0010\ldots, \ldots\}$. Then, since our penetrator bound $I^{\mathsf{c}}$ is closed under penetrator actions and this flawed $\mathbf{X}$-strand allows arbitrary use of XOR, all of $\mathsf{X}$ is within the penetrator's bound. But then there are no XOR keys that are guaranteed to remain secret, for $(S \cap \mathbf{X}) \subset (I \cap \mathbf{X}) = \emptyset$. This bound on the penetrator is much too large, as it's trivially true that the set of keys known by the penetrator is contained in the set of all keys. The ideal model assumes that if the penetrator can possibly deconstruct a message and obtain a secret value, then that message is dangerous. This is a good model for encryptions and concatenations, because there are a small number of possible deconstructions, all of which terminate after a certain number of steps, and if any one of them leads to a secret value, the penetrator could feasibly find it. However, XOR products can be deconstructed

in many ways, and may continue for an arbitrary number of steps. Choosing a set of basis keys really does the penetrator no more good that making random guesses of keys, even though a secret key can be constructed as some product of the basis. The difficulty is in knowing which keys to include in the product, and if the penetrator does manage to calculate a secret key from the basis, then he must have made the correct decision of whether or not to include each of the $l$ basis keys in the product. This is equivalent to guessing correctly a bit string of length $l$, which is a random guess in $\mathsf{X}$.

Therefore, the penetrator's actions are bounded in a way that treats penetrator guesses as such. Notice that if at least one of $K, K'$ is chosen randomly according to a uniform distribution, then the product $K \oplus K'$ also has a uniform distribution. This is because the mapping $K' \mapsto K \oplus K'$ is a bijection between finite, discrete spaces, so the uniform probability distribution of $K'$ is transfered to the product. Therefore, if the penetrator calculates an XOR product with at least one random key, the result is again random. Thus, any such XOR Tree is effectively just a guess, and the key is contained in the set $\mathsf{X}_{\mathcal{P}}$ of penetrator guesses of XOR keys. The only interesting use of XOR occurs when the penetrator acts only on keys emitted by the regular participants.

### *2.1.3 XOR Ideals*

The following construction of ideals mimics that of ideals in unstructured protocols, taking into account the additional structure of XOR. Recall that the purpose of constructing an ideal is to bound penetrator activity away from secret data.

Denote the subset of XOR keys known by the penetrator as $\mathsf{X}_{\mathcal{P}} = \mathsf{X} \cap \mathsf{A}_{\mathcal{P}}$; the penetrator sets $\mathsf{K}_{\mathcal{P}}$ and $\mathsf{T}_{\mathcal{P}}$ are previously defined. Suppose that for a certain protocol there is a finite secret set $S \subset \mathsf{T} \cup \mathsf{K}$. The secret XOR keys are $S_{\mathsf{X}} = S \cap \mathsf{X}$, and we assume that $S_{\mathsf{X}}$ is an independent set. Therefore, $S_{\mathsf{X}} = \{K_1, K_2, \ldots, K_n\}$

where the $K_i$ form a basis of $\langle S_{\mathsf{X}} \rangle$. We wish to find a set of messages that bound the penetrator actions.

To that end, consider the set $H$ of all *even* keys in $\langle S \rangle$,

$$\begin{aligned} H &= \mathrm{Even}(\S) \\ &= \left\{ K \in \langle S_{\mathsf{X}} \rangle \;\middle|\; K = \bigoplus_{k=1}^{2m} K_{i_k}, 1 \leq m \leq \binom{n}{2}, i_k \in [1, \ldots, n] \right\}. \end{aligned}$$

The $i_k$ need not be distinct in this representation, for repeated keys will cancel in pairs and leave an even number of distinct keys. Every key in $\mathrm{Even}(\S)$ can be written as the product of an even number of distinct keys in $S_{\mathsf{X}}$. The even keys form a subgroup/subspace of $\langle S_{\mathsf{X}} \rangle$, since the repeated keys cancel in pairs and therefore preserve the parity of the number of basis keys in a product. An independent basis of $\mathrm{Even}(\S)$ is the set $\{K_1 \oplus K_i \mid 2 \leq i \leq n\}$. The complement of $\mathrm{Even}(\S)$ in $\langle S_{\mathsf{X}} \rangle$ is the odd keys $\mathrm{Odd}(\S) = \langle S_{\mathsf{X}} \rangle \setminus \mathrm{Even}(\S)$. Every key in $\mathrm{Odd}(\S)$ can be written as the product of an odd number of distinct keys from $S_{\mathsf{X}}$.

The keys in $\mathrm{Even}(\S)$ and $\mathrm{Odd}(\S)$ are equivalent to subsets of keys chosen from $S_{\mathsf{X}}$ of even and odd size, respectively. Recall the well-known fact from combinatorics that there are an equal number of even and odd subsets for any finite set. Therefore, both $\mathrm{Even}(\S)$ and $\mathrm{Odd}(\S)$ contain exactly half of the keys in the span of $S$, i.e. $|\mathrm{Even}(\S)| = |\mathrm{Odd}(\S)| = |\langle S_{\mathsf{X}} \rangle| / 2$. This can also be seen by considering the basis for $\mathrm{Even}(\S)$ given above. There are exactly $n - 1$ independent basis keys, which is one less than the number of basis keys for $\langle S_{\mathsf{X}} \rangle$. Since $\mathrm{Even}(\S)$ is a subspace of the vector field $\langle S_{\mathsf{X}} \rangle$, and the underlying field of binary bits has two elements, the subspace of even keys is half the size of the overall space.

Note the following properties of $\mathrm{Even}(\S)$ and $\mathrm{Odd}(\S)$, which also serve as an inductive definition of the sets:

- $K_i \in \mathrm{Odd}(\S)$ for $i \in [n]$, so $S_{\mathsf{X}} \subset \mathrm{Odd}(\S)$;
- If $K, K' \in \mathrm{Odd}(\S)$, then $K \oplus K' \in \mathrm{Even}(\S)$;

- If $K, K' \in \mathrm{Even}(\S)$, then $K \oplus K' \in \mathrm{Even}(\S)$;
- If $K \in \mathrm{Even}(\S)$ and $K' \in \mathrm{Odd}(\S)$, then $K \oplus K' \in \mathrm{Odd}(\S)$.

These properties can be shown through parity arguments on the number of distinct basis keys in the representation of $K$ and $K'$. For example, if $K, K' \in \mathrm{Odd}(\S)$, then both keys are the product of an odd number of distinct keys. In their product $K \oplus K'$, there are an even number of total keys, and again the parity is preserved when key pairs are canceled. For example, $(K_1 \oplus K_2 \oplus K_3) \oplus (K_2) = K_1 \oplus K_3 \in \mathrm{Even}(\S)$.

The product $K \oplus K' \in \mathrm{Odd}(\S)$ if and only if exactly one of $K$ and $K'$ is also in $\mathrm{Odd}(\S)$. Therefore, if the penetrator can learn only keys in $\mathrm{Even}(\S)$, then no keys in $\mathrm{Odd}(\S)$ can be calculated, including the secret keys $S_{\mathsf{X}}$. This means that a protocol is secure if the regular participants only emit keys whose span is disjoint from $\mathrm{Odd}(\S)$. This condition prevents the honest parties from emitting pairs of messages like $K \oplus K_1 \oplus K_2$ and $K \oplus K_1$, for the product of the messages is the secret key $K_2$. However, requiring the messages to be outside of $\mathrm{Odd}(\S)$ is more restrictive than necessary, for it is admissible for the penetrator to learn a triple product such as $K_1 \oplus K_2 \oplus K_3$ as long as he cannot use it to learn a secret key. The important condition is that the span of the messages from regular participants is disjoint from the secret set $S_{\mathsf{X}}$. It is usually easiest to use $\mathrm{Odd}(\S)$ and $\mathrm{Even}(\S)$, for then the relationships between the keys only involve simple parity arguments.

Recall that an ideal represents the set of dangerous messages in a protocol. Suppose for simplicity that we have an XOR protocol in which all of the secret values are XOR keys, so that $S_{\mathsf{X}} = S$, and the span of messages emitted by regular participants is contained in $\mathrm{Even}(\S)$. Define the ideal $I = I_{\mathsf{k}}[\langle S_{\mathsf{X}} \rangle \setminus \mathrm{Even}(\S)]$ where $\mathsf{k} = \mathsf{K} \setminus (\langle S_{\mathsf{X}} \rangle \setminus \mathrm{Even}(\S))$. Then, since $I$ is an ideal, its complement is closed under all of the old penetrator actions. Since the penetrator can only use XOR on the keys emitted by regular participants, and all of these keys are in XOR keys

in $\mathrm{Even}(\S) \subset I^{\mathsf{c}}$, the complement is also closed under XOR. Therefore, the ideal construction shows the secrecy of $S$.

This is shown more carefully in the following proof of closure in general ideals.

**Theorem.** *Let $H$ be the span of messages emitted by regular participants, and $\mathsf{k} = \mathsf{K} \setminus (\langle S \rangle \setminus H)$. Construct the ideal $I = I_{\mathsf{k}}[\langle S \rangle \setminus H]$; its complement $I^{\mathsf{c}}$ is then closed under penetrator actions.*

*Proof.* The definition of $H$ is analogous to $\mathrm{Even}(\S)$, and $H^{\mathsf{c}}$ to $\mathrm{Odd}(\S)$, however, $H$ may contain keys and nonces not in $\mathsf{X}$. The complement is taken in the texts and keys, so $H^{\mathsf{c}} = \mathsf{K} \cup \mathsf{T} \setminus H$. Also, the XOR keys in $H$ are not necessarily all in $\langle S_{\mathsf{X}} \rangle$, as a regular participant may send an XOR key in unencrypted form. Recall that if $S$ contains some value not in $\mathsf{X}$, then $\langle S \rangle$ also contains this value.

Since $I$ is an ideal, $I^{\mathsf{c}}$ is still closed relative to $\mathbf{C}, \mathbf{S}, \mathbf{E}$ and $\mathbf{D}$-strands. It remains to show closure under $\mathbf{X}$-strands. XOR is only defined on the set $\mathsf{X}$, so we need only consider XOR keys. But the $\mathbf{X}$-strand is restricted further, and operates only on keys learned from regular participants, so that XOR is actually used only keys from $H$. By definition, $H \subset I^{\mathsf{c}}$, and $H$ is closed. Now we show that if the node $p$ is an entry point to $I$ on an $\mathbf{X}$-strand, then $p$ has negative sign. The trace $\langle -K,\ -K',\ +K \oplus K' \rangle$ is defined only when $K, K' \in H$, and since $H$ is closed, $K \oplus K' \in H$. Thus, there are no entry points to $I$ on any XOR strand, and $I^{\mathsf{c}}$ is closed under XOR, and all other penetrator actions. □

In this theorem, $H$ may contain some of the secret keys, and the closure has been proved for ideals in general, regardless of whether they truly represent the dangerous messages. To prove secrecy, we need a corollary

**Corollary.** *Let $H$ be defined as before. If $H \cap S = \mathsf{A}_{\mathcal{P}} \cap S = \emptyset$, then the protocol preserves the secrecy of $S$.*

*Proof.* Construct the ideal $I$ as in the theorem. Then $I^{\mathsf{c}}$ is closed under penetrator actions, and since he only has access to terms from $H \subset I^{\mathsf{c}}$ and $\mathsf{A}_{\mathcal{P}} \subset I^{\mathsf{c}}$, every penetrator node has term in $I^{\mathsf{c}}$, and is disjoint from $S \subset I$. □

This result holds for any bundle of nodes, and thus in a protocol that satisfies the corollary conditions, there is no attack that compromises the secrecy of $S$.

### *2.1.4 Other key subspaces*

The ideal construction above is a general method for proving the secrecy of protocols, but it is easier to work with even and odd keys, for then we can use parity arguments as well as working in the vector space. Fortunately, a clever change of basis allows us to use even and odd sets.

For example, consider a protocol for distributing a fresh pair of keys. Two principals $A$ and $B$ share the keys $K_1, K_2$ and use them to communicate in some cryptosystem. To prevent the penetrator from guessing one of the keys through brute force methods, the principals wish to securely exchange a pair of new keys $K_3, K_4$. The secret set is thus $S_{\mathsf{X}} = \{K_1, K_2, K_3, K_4\}$, and we assume that this is an independent set. Disregarding authentication for the moment (authentication steps can easily be added to the end of a key exchange), the following two messages are sent:

$$A \to B: \quad K_1 \oplus K_2 \oplus K_3$$
$$B \to A: \quad K_1 \oplus K_2 \oplus K_4$$

Here we have $A$ choose the new key $K_3$, and $B$ choose $K_4$. Since they both know $K_1$ and $K_2$, each can calculate the key chosen by the other. Neither message is a secret key, and their product $K_3 \oplus K_4$ isn't either, so the penetrator clearly cannot learn a secret key. However, to prove the secrecy, we construct an ideal around the complement of the span of the emitted keys. This ideal is larger than one constructed around $\mathrm{Odd}(\S)$, and more complicated.

Consider what happens if we use the basis $S' = \{K_{1,2}, K_2, K_3, K_4\}$, where $K_{1,2}$ is shorthand for $K_1 \oplus K_2$. Since these keys form an independent set, the set of products contains every key in $\langle S_{\mathsf{X}} \rangle$. The protocol can then be written as:

$$A \to B : \quad K_{1,2} \oplus K_3$$
$$B \to A : \quad K_{1,2} \oplus K_4$$

Both of the transmitted messages are now in $\mathrm{Even}(S')$, so we can conclude the secrecy of $S'$ using the ideal $I_{\mathsf{k}}[\mathrm{Odd}(S')]$. The secret key $K_1$ isn't in $S'_{\mathsf{X}}$ though, or even in $\mathrm{Odd}(S')$, since $K_1 = K_{1,2} \oplus K_2$. However, all of the secret keys are in the complement of $\mathrm{Even}(\{K_{1,2}, K_3, K_4\})$. We observed earlier that it is easier to work with even keys than an arbitrary subspace. It appears to be true that any subspace is in fact the even keys for some set of independent vectors. However, it seems to difficult to find these vectors, and it is yet unclear how to prove even the existence of such vectors.

Instead, we prove the following:

**Claim.** *The $n$-dimensional vector space $\langle S_{\mathsf{X}} \rangle$ with basis $\{K_1, \ldots, K_n\}$ has exactly one subset $H$ of dimension $n-1$ that satisfies $K_i \notin H$ for each $i \in [1, n]$. In particular, $H = \mathrm{Even}(\S)$.*

*Proof.* First we show that a closed subspace $H \subset \mathsf{X}$ is either a proper subset of $\mathrm{Even}(\S)$, or $H$ contains equal numbers of even and odd keys. The first case occurs whenever $H$ is a subspace of $\mathrm{Even}(\S)$. Now suppose that there is some odd key $K_O \in H \cap \mathrm{Odd}(\S)$. Then the mapping $\phi(K) = K \oplus K_O$ gives a bijection between $H \cap \mathrm{Even}(\S)$ and $H \cap \mathrm{Odd}(\S)$, so the finite sets of even and odd keys in $H$ are equal in size.

Now suppose that $H$ has dimension $n-1$; we show that $H = \mathrm{Even}(\S)$. Consider the partition of $\langle S_{\mathsf{X}} \rangle$ into classes of the form $\{K, K \oplus K_1\}$. The closed subspace $H$ contains at most one element from any of these classes, since the secret key $K_1$

is calculable from $K$ and $K \oplus K_1$. The underlying field has two elements, so $H$ contains half of the elements of $\langle S_{\mathsf{X}} \rangle$, and therefore exactly one element from each of the classes. This includes the classes of the form $\{K_i, K_i \oplus K_1\}$ where $i \neq 1$. The key $K_i$ is secret, so $K_i \notin H$ and instead $K_i \oplus K_1 \in H$. The set $\{K_i \oplus K_1 \mid 2 \leq i \leq n\}$ is a basis for $\mathrm{Even}(\S)$, and thus $\mathrm{Even}(\S) \subset H$. However, since $\mathrm{Even}(\S)$ also has dimension $n-1$, it must be true that $H = \mathrm{Even}(\S)$. □

This result on subspaces of the XOR space also follows from a more general theorem on finite fields:

**Theorem.** *The $n$-dimensional vector field over the finite field of $k$ elements (with $k$ prime), $V = \mathbb{F}_k^n$, with basis $B = \{B_1, \ldots, B_n\}$, has exactly $(k-1)^{n-1}$ subspaces of dimension $n-1$ that are disjoint from $B$.*

*Proof.* Define a bilinear product for pairs of vectors $x, y \in V$ by $\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$. Since $k$ is prime, the field $\mathbf{F}_k \cong \mathbb{Z}/\mathbb{Z}_k$. For some $x \in V$, with $x$ not equal to the zero vector, define the set $H_x = \{y \mid \langle x, y \rangle = 0\}$. Note that it's possible that $x \in H_x$; for example, if $x$ has exactly $k$ 1's and $n-k$ 0's. By linearity, $H_x$ is a closed subspace of $V$. The dimension of $H_x$ is at most $n-1$, since if $x$ is nonzero in coordinate $i$, then $\langle x, e_i \rangle$, where $e_i$ is the standard basis element in the $i$th coordinate. We now construct an independent basis to show that the dimension is in fact $n-1$. Again, find some $i$ such that $x_i \neq 0$. Then for all $j \neq i$, there are two cases. If $x_j = 0$, then let $A_j = e_j \in H_x$, since the only nonzero coordinate in $A_j$ is multiplied by the zero in $x$. If $x_j \neq 0$, then let $A_j = (x_i)^{-1} e_i + (-x_j)^{-1} e_j$, where the inverses are uniquely found in the field. This choice gives $\langle x, A_j \rangle = 1 - 1 = 0$. These $n-1$ vectors are independent, since $A_j$ is nonzero in coordinate $j$ and zero in coordinate $l$ for all $l \neq j, i$. Thus $H_x$ has dimension $n-1$.

For simplicity, make a change of coordinates in $V$ so that $B$ is the standard basis, $B = \{e_1, \ldots, e_n\}$. Such a change of coordinates preserves the independence of subspaces, so the desired count is unaffected. We now wish to characterize the

subspaces of $V$ that are disjoint from the standard basis. Note that if $x$ is zero in some coordinate $i$, then $\langle x, e_i \rangle = 0$, and $e_i \in H_x$. Thus the $H_x$ we seek are generated by vectors $x$ that are nonzero in every coordinate. There are $(k-1)^n$ such vectors in $V$. However, by the linearity of the vector product, $H_x = H_{2x} = \cdots = H_{(k-1)x}$. Since the characteristic of the field is prime, each of these multiples are distinct for any $x$. Thus we group the nonzero vectors into $(k-1)^{n-1}$ classes, giving an upper bound on the number of distinct subspaces of dimension $n-1$ that satisfy the given property. It remains to show that the $H_x$ is unique for the class generated from $x$.

Suppose that $H_x = H_y$ for some $x$ and $y$ that are nonzero in each coordinate. We show that $x$ is a multiple of $y$. There exists some $g \in [1, k-1]$ such that $x_1 = gy_1$, since both are nonzero. Then, for any $i \in [2, k-1]$, the vector $v_i = (y_1)^{-1}e_1 + (-y_i)^{-1}e_i \in H_y$, since the product $\langle y, v_i \rangle = 1 - 1 = 0$. Thus $v_i \in H_x$ as well, so $\langle x, v_i \rangle = 0$. Expanding the product, $0 = x_1(y_1)^{-1} + x_i(-y_i)^{-1} = gy_1(y_1)^{-1} - x_i(y_i)^{-1} = g - x_i(y_i)^{-1}$. This implies that $x_i = gy_i$ for all $i$, and thus $x = gy$, so each class of vectors generates a unique subspace. □

The vector field that arises due to the XOR structure is over the two-element field, so the theorem states that there is exactly 1 $(n-1)$-dimensional subspace disjoint from the basis.

We now have a good understanding of how to prove secrecy for XOR protocols. However, authentication of XOR protocols proves to be a thornier issue, as illustrated by the involved analysis of the TMN protocol in the next section.

### *2.1.5 TMN Protocol*

Recall the TMN protocol, seen previously in section 2.1:

$$
\begin{aligned}
A \rightarrow S: &\quad \{m_1\ B\ K_a\}_{K_{AS}} \\
S \rightarrow B: &\quad A\text{req} \\
B \rightarrow S: &\quad \{m_3\ A\ K_b\}_{K_{BS}}
\end{aligned}
$$

$$S \rightarrow A: \quad K_a \oplus K_b$$
$$A \rightarrow B: \quad \{A\}_{K_b}$$
$$B \rightarrow A: \quad \{A\ B\}_{K_b}$$

where $K_a, K_b$ are XOR keys in $\mathsf{X}$. The cryptosystem is symmetric, and the keys $K_{AS}$ and $K_{BS}$ are shared between the server and a principal. The messages $m_1$ and $m_3$ contain identifying information to prevent certain types of replay attacks. A new key $K_a$ is by $A$ each time the protocol is run to prevent a brute-force attack by the penetrator, for if the same key were used repeatedly the penetrator could spend a long time analyzing a group of messages.

An infiltrated strand space $\Sigma_{TMN}, \mathcal{P}$ is a *TMN* space if $\Sigma_{TMN}$ is the union of four types of strands:

1. Penetrator strands $s \in \mathcal{P}$.

2. "Initiator strands" $s \in \text{Init}\,[A, B, K_a, K_b]$ with trace:

$$\left\langle +\{m_1\ B\ K_a\}_{K_{AS}}, -K_a \oplus K_b, +\{A\}_{K_b}, -\{A\ B\}_{K_b} \right\rangle$$

   where $A, B \in \mathsf{T}_{name} and K_a, K_b \in \mathsf{X}$. The initiator $A$ is the principal associated with these strands.

3. "Responder strands" $s \in \text{Resp}\,[A, B, K_a, K_b]$ with trace:

$$\left\langle -Areq, +\{m_3\ A\ K_b\}_{K_{BS}}, -\{A\}_{K_b}, +\{A\ B\}_{K_b} \right\rangle$$

   The responder $B$ is the principal associated with these strands.

4. "Server strands" $s \in \text{Serv}\,[A, B, K_a, K_b]$ with trace:

$$\left\langle -\{m_1\ B\ K_a\}_{K_{AS}}, +Areq, -\{m_3\ A\ K_b\}_{K_{BS}}, +K_a \oplus K_b \right\rangle$$

The secret values for the TMN protocol are $S = \{K_a, K_b, K_{AS}, K_{BS}\}$. The subset $H$ is given by the span of the XOR terms emitted by regular participants, so $H = \{K_a \oplus K_b\} = \mathrm{Even}(\S)$, since $\mathrm{Even}(\S)$ contains only XOR keys. Define $\mathsf{k} = \mathsf{K} \setminus H^{\mathsf{c}} = \mathsf{K} \setminus \{K_a, K_b, K_S^{-1}, K_{AS}, K_{BS}\}$. Then the ideal $I = I_{\mathsf{k}}[H^{\mathsf{c}}]$ proves the secrecy of the TMN protocol, since all of the terms emitted by regular participants are in $I^{\mathsf{c}}$, and $I$ contains no publicly known terms.

This version of the TMN protocol has an initiator guarantee for a responder strand. The initiator knows that $K_b$ is a secret key, and when he receives the message $\{A\ B\}_{K_b}$, he knows that it must have originated on the final node of a responder strand in $\mathrm{Resp}[A, B, K_b]$. If we assume that $K_b$ uniquely originates, then the responder strand is unique.

To prove a responder guarantee for the initiator, we start with the message $\{A\}_{K_b}$ and trace its path, hopefully ending at the desired initiator strand. We know by the secrecy of $K_b$ that this message originated on an initiator strand in $\mathrm{Init}[A, X, K_a, K_b]$, where $X$ is some principal. We want to show that $X = B$, and it seems that this may be done by considering the two messages at the beginning of the strand, $\{m_1\ X\ K_a\}_{K_{AS}}$ and $K_a \oplus K_b$. The key $K_a$ must be the same in both of these messages, and since the server will only send out the product $K_a \oplus K_b$ when the principal names match, we conclude that $A$ intended to talk to $B$, so the initiator strand has the correct form. However, this argument requires the use of a server strand, and there is a flawed server strand guarantee for both the initiator and responder. In the following replay attack, the initiator $A$ and responder $B$ both participate in two concurrent runs of the protocol, and the penetrator exploits the XOR structure to impersonate the server. The two runs are differentiated by subscript.

First, the penetrator copies the messages sent to the server by $A_1$ and $B_1$, and intercepts the key emitted by the server:

$A_1 \rightarrow S$: $\{m_1\ B\ K_a\}_{K_{AS}}$
$S \rightarrow B_1$: $A$req
$B_1 \rightarrow S$: $\{m_3\ A\ K_b\}_{K_{BS}}$
$S \rightarrow P(A_1)$: $K_a \oplus K_b$.

Then he performs the same actions in the second run of the protocol:

$A_2 \rightarrow S$: $\{m_1\ B\ K'_a\}_{K_{AS}}$
$S \rightarrow B_2$: $A$req
$B_2 \rightarrow S$: $\{m_3\ A\ K'_b\}_{K_{BS}}$
$S \rightarrow P(A_2)$: $K'_a \oplus K'_b$

The penetrator now uses the messages he has read to present a third protocol run to the server, and learn another key product:

$P(A) \rightarrow S$: $\{m_1\ B\ K'_a\}_{K_{AS}}$
$S \rightarrow P(B)$: $A$req
$P(B) \rightarrow S$: $\{m_3\ A\ K_b\}_{K_{BS}}$
$S \rightarrow P(A)$: $K'_a \oplus K_b$

Finally, the penetrator uses the product $K'_a \oplus K_b$ to establish communication between $A_2$ and $B_1$. He also calculates $K_a \oplus K'_b = (K_a \oplus K_b) \oplus (K'_a \oplus K'_b) \oplus (K'_a \oplus K_b)$, to complete the protocol between $A_1$ and $B_2$:

$P(S) \rightarrow A_1$: $K_a \oplus K'_b$
$A_1 \rightarrow B_2$: $\{A\}_{K'_b}$
$B_2 \rightarrow A_1$: $\{B\}_{K'_b}$
$P(S) \rightarrow A_2$: $K'_a \oplus K_b$

$A_2 \rightarrow B_1$: $\{A\}_{K_b}$

$B_1 \rightarrow A_2$: $\{B\}_{K_b}$

But the communication between $A_2$ and $B_1$ violates server authentication, since there was no server strand that generated the product $K'_a \oplus K_b$.

For this attack to succeed, there were several server strands, and the penetrator couldn't do this alone. However, none of the strands had the form that the principals would expect. It could also be problematic that the order of protocol runs was swapped between the initiator and responder, but protocols usually operate independently. The communication established between $B_1$ and $A_2$ is still secure.

It isn't clear how to proceed in the proof of a responder guarantee without knowing the exact form of the server strands that are present in some protocol run. To help us understand the interactions between the server and the other principals, we consider a simpler protocol similar to TMN without a server. The responder $B$ acts as the key distributor:

$A \rightarrow B$ : $\{Areq\ K_a\}_{K_{AB}}$

$S \rightarrow B$ : $K_a \oplus K_b$

$A \rightarrow B$ : $\{A\}_{K_b}$

$B \rightarrow A$ : $\{A\ B\}_{K_b}$

Here we assume that $A$ and $B$ already share the symmetric key $K_{AB}$. This protocol maintains the secrecy of $S = \{K_a, K_b, K_{AB}\}$ by similar arguments to those made for the TMN protocol, and as before, the initator has a full guarantee on the responder.

Now examine what happens when the penetrator tries to impersonate the responder/server as before. The only possible replay is for the penetrator to send a request from $A$ to $B$ multiple times. In this protocol, the penetrator doesn't have as much control over which keys are combined by the server, for $B$ chooses a new key for each run of the protocol. For example, if the penetrator repeatedly sends the message $\{Areq\ K_a\}_{K_{AB}}$, the responder will emit the products $K_a \oplus K_b, K_a \oplus$

$K'_b, K_a \oplus K''_b, \ldots$. Each of the keys from the responder are randomly chosen and therefore the key products learned by the penetrator are also random and most likely independent. Any additional product containing $K_a$ that the penetrator calculates is the product of at least four keys, so he cannot imitate a regular party unless there are other dependencies in the keys. Since any such dependencies occur randomly, the penetrator is effectively reduced to guessing when he tries to exploit his knowledge. This is a limited capability compared to the TMN protocol.

The problem we had when trying to prove a responder guarantee for the TMN protocol was that the responder had no way of knowing whether the expected server strand was actually present. However, in this protocol, the responder acts as the server. Therefore when the responder receives the message $\{A\}_{K_b}$, he knows that he sent out $K_a \oplus K_b$ and that the penetrator could not have constructed this message. Therefore, the initiator strand is in Init$[A, B, K_a, K_b]$, as needed for the authentication guarantee.

It is possible to analyze the penetrator's actions more formally when the keyspace is large compared to the number of principals and the maximum number of protocol runs we will allow. First we change our approach to randomly chosen keys. The random choice of $K_b$ is viewed as the choice of $K_b$ from some finite set of keys representing the possible random choices. This is similar to the way penetrator guesses are modeled. Define the sets $P^Q$ as the set of keys chosen by $P$ as the initiator chooses for communication with $Q$ as the responder, and similarly, $Q_P$ as the set of keys chosen by $Q$ as the responder for communication with $P$ as the initiator for all pairs (possibly identical) of principals $Q$ and $P$.

Since the keyspace is larger compared to these sets, we can safely assume that randomly chosen keys are all independent. We make the further assumption that the sets of protocol keys defined above are also independent, so that $Q^P \cap C^D \neq \emptyset$ if and only if $Q = C$ and $P = D$. This condition also holds for the initiator and responder sets, so that $Q^P \cap C_D = \emptyset$, even for identical sets of principals.

Using this notation, we can place further restrictions on the simplified TMN protocol. In the initiator message $\{Areq\ K_a\}_{K_{AB}}$, the key $K_a$ must be chosen from $A^B$, and the responder must choose some $K_b \in B_A$ to calculate the product $K_a \oplus K_b$. Therefore, the key products emitted by a responder strand in Resp$[A, B, K_a, K_b]$ are always in $A^B \oplus B_A$, where the XOR product of two sets is defined in the obvious way: if $X, Y \subset \mathsf{X}$, then $X \oplus Y = \{x \oplus y \mid x \in X, y \in Y\}$.

Note that $A^B \oplus B_A \subset \text{Even}(A^B \cup B_A) \subset \text{Even}(\S)$ for all pairs of principals, so every key that the penetrator can learn is an even products in $S_{\mathsf{X}}$. In fact, the penetrator's knowledge can be bounded more tightly, by $\mathsf{X}_{\mathcal{P}} \cap \langle S_{\mathsf{X}} \rangle \subset \left\langle \bigcup_{i,j} \text{Even}\left( {A_i}^{A_j} \oplus {A_j}_{A_i} \right) \right\rangle$.

To prove the responder's guarantee, we consider a strand $r \in$ Resp$[A, B, K_a, K_b]$. We want to show that if the message $\{A\}_{K_b}$ received by $r$ originated on a strand $s \in$ Init$[A, X, K, K_b]$, then $X = B$ and $K = K_a$, so that the expected initiator strand is in the bundle. Since $K_b$ is secret, $A$ must have calculated it from the key received in the second step of the protocol. He used the key $K$ to undo the XOR, so $K$ must satisfy $K \oplus (K_a \oplus K_b) = K_b$. This implies that $K \oplus K_a = K_{\mathsf{id}}$, so $K = K_a$.

Now suppose that $X \neq B$. Then since $K_a \in A^X$ and $K_b \in B_A$, the product $K_a \oplus K_b$ isn't calculated by a server strand. The only other possibility is that the key was generated through penetrator actions. However, any key generated by the penetrator must have an even number of keys from each set of the form $P^Q \cup Q_P$, and the product $K_a \oplus K_b$ contains one key from each of $A^X \cup X_A$ and $A^B \cup B_A$. Therefore the message $K_a \oplus K_b$ couldn't have been generated by any party if $X \neq B$, and it therefore must be that $X = B$. This completes the proof of the authentication guarantee for the responder, since he can deduce that the initiator strand $s$ is in Init$[A, B, K_a, K_b]$.

At no point in this analysis did we use the fact that the responder acts as the server. In the TMN protocol the product keys emitted by the server are also in sets of the form $A^B \cup B_A$, so this approach also proves the responder's guarantee for the TMN protocol. In fact, this works for any protocol in which the party that emits

the XOR product requires verification of identities. If the emitted XOR product has more than two keys, then the group structure is more complicated. In this case, we again use the span of all the possible products and show that only the desired principal could have generated a certain product. Also, if one party selects several keys for use with another party in a protocol, there should be a key set associated with each distinctly chosen key. For example, if $A$ chooses a random key $K_a$ and sends it to $B$, and at a later point in the protocol chooses another random key $K'_a$, then $K_a \in A^B$ and $K'_a \in \left(A^B\right)'$.

The problem with this method is that it requires either a very large keyspace or a limited amount of action by the principals. As we saw when constructing XOR ideals (section 2.1.3), considering all possible products overestimates the abilities of the penetrator. Thus, even when the key sets are possibly dependent, it is unlikely that the penetrator will discover the dependencies (again, it reduces to a lucky guess), so the analysis is valid.

We would like to find a more general method for proving the authentication properties of XOR protocols. For unstructured protocols, it was enough to trace a path to the origination point of a received message; however, the XOR structure makes it more difficult to do this precisely.

### *2.1.6 XOR on larger sets*

In the preceding sections, we have discussed protocols in which XOR is used only on a subset of keys. We now show that these results also apply to protocols in which both keys and nonces have XOR structure. Suppose that the set of XOR terms can contain both keys and nonces, so that $\mathsf{X} \subset \mathsf{K} \cup \mathsf{T}$. The set of texts contains both names and nonces; however, it is unreasonable to include the names in the XOR structure. Principal names are all the same data type and contain identifying bits in their representation and are therefore not be part of a closed XOR

set. Furthermore, since names are public and may be large in number, their XOR closure could be a significant portion of all XOR strings.

When $\mathsf{X} \subset \mathsf{K} \cup \mathsf{T}$, with principal names excluded, then we construct an ideal just as before, except for this case, the additional XOR structure is included. Recall that an ideal contains the potentially dangerous messages, and is constructed from the complement of the closure of the emitted messages. This closure now contains products of nonces and keys. We have already proved that the complement of an ideal is closed under penetrator actions when there are XOR keys. Therefore, even if each nonce were a key, the ideal complement would be closed. The penetrator has fewer available actions with the nonces, so the bound of the ideal still holds, and proves the secrecy of $S$. Thus, when there are both XOR nonces and keys, we assume in our analysis that the two data types are indistinguishable.

However, the two sets are truly identical only in certain situations. If at least one of $\mathsf{K_X}$ and $\mathsf{T_X}$ (the XOR keys and nonces, respectively) are not closed under XOR, then the sets are free to intersect in any way. In this case, it is impossible to predict the data type of a product, and in a sense, the keys and nonces are indistinguishable again. If both $\mathsf{K_X}$ and $\mathsf{T_X}$ are closed under XOR, then one of the sets contains the other, and at least one of the sets is equal to $\mathsf{X}$ itself. Suppose that $\mathsf{K_X} \not\subset \mathsf{T_X}$. Then there is some $K \in \mathsf{K_X}$ such that $K \notin \mathsf{T_X}$. Thus, for any $T \in \mathsf{T_X}$, the identity $T \oplus (K \oplus T) = K$ implies that $K \oplus T \notin \mathsf{T_X}$, since $\mathsf{T_X}$ is closed and $\mathsf{K} \notin \mathsf{T_X}$. Therefore, for each $T \in \mathsf{T_X}$, both $K, K \oplus T \in \mathsf{K_X}$ so $T \in \mathsf{K_X}$ as well, and $\mathsf{T_X} \subset \mathsf{K_X}$. One of the sets must be equal to $\mathsf{X}$ itself, since $\mathsf{X}$ is just the union of the two sets.

The general problem of XOR on an arbitrary closed set of unencrypted terms is more difficult. It isn't clear what assumptions are needed so that the ideal construction can still be used to prove secrecy. However, this problem is important, as it would allow the analysis of protocols in which XOR is used as the cryptosystem.

# Chapter 3

# Conclusion

We have shown how to extend the strand space machinery to analyze XOR protocols, using extensions of previous results that take into account the structure of XOR.

The most important construction for analyzing unstructured protocols is the bound on the penetrator's actions that arises from ideal complements. A carefully constructed bound is used to prove that in a well-designed protocol the penetrator is unable to learn certain secret values, no matter how cleverly he proceeds. That such a bound exists is proved by modeling penetrator actions as algebraic operations on the message space, and then using honest ideals to construct sets that are closed relative to these operations. Ideals are used directly to prove the secrecy of nonces and both randomly chosen and private keys. Then the secrecy of certain terms is used to prove authentication guarantees, since upon receiving a secret term in encrypted form, a principal can deduce which party could have possibly sent such a message. In a good protocol, that party must be the expected principal.

In XOR protocols, the new operation adds additional algebraic structure to the messages. However, the structure is that of a well understood group or vector space, and the penetrator actions are again bounded by taking a closed subgroup generated from the messages emitted by regular participants. Again, an ideal construction based on such a subgroup is used to prove secrecy in an XOR protocol. Authentication is more difficult to prove, since it isn't always clear how a message may have been manipulated with XOR. There is a subtle challenge in bounding the penetrator's actions, as it is difficult to distinguish between random guessing and

more knowledgeable manipulations.

### *3.1 Future Work*

There are many useful directions in which this research can continue. The study of XOR protocols is incomplete, as we do not have general methods for proving authentication results. Another important extension is the study of protocols in which XOR is used as the symmetric encryption algorithm. It may be worthwhile to use the machinery we have developed to analyze such protocols, keeping in mind that every message is now a product.

Protocols that use modular functions instead of XOR to aid in distributing fresh keys are also widely used. For example, in the well known Diffie-Helman key-exchange protocol, two regular participants choose nonces $x$ and $y$, and transmit $g^x \mod p$ and $g^y \mod p$, where $g$ is a generator of the multiplicative group modulo the prime $p$. Both principals can then calculate $g^{xy}$, and presumably the penetrator can only obtain this value by either calculating the discrete logarithm or exploiting a protocol flaw. The goal in studying such protocols would be to understand the group structure and contain the penetrator actions as we did with XOR in this paper. Through preliminary investigations of this problem, we believe that for certain choices of the base prime $p$, most of the penetrator actions are equivalent to random guesses, and therefore the ideal construction should be applicable.

Many of the assumptions of the strand space model and other similar systems are based on the fact the message space is very large, and random guesses of keys and nonces are very unlikely to be correct. We made similar assumptions when considering protocols with algebraic structure in the message space. These assumptions depend on the "sparseness" of the additional structures, so that we could reasonably assume that distinct random values are independent in a useful sense. We would like to see rigorous justification of the assumptions. One possible

approach is using nonstandard analysis to prove asymptotic results. It is easier to consider guesses when the spaces are infinitely large. If useful results could be shown in this context, then the overspill property of nonstandard sets would guarantee the existence of large, finitie spaces for which the results also hold.

Finally, our method for extending strand spaces relies primarily on finding closed subsets of messages and using them to bound the penetrator's actions. This should be applicable to a variety of different algebraic structures, and it is natural to consider message algebras with arbitrary identities. However, given the complexities that arose in ideal construction even in the simple case of XOR, the general case seems to require a more powerful method.

# Bibliography


[1] F. Javier Thayer Fabrega, Joshua D. Guttman, and Jonathan C. Herzog. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, (7):191–230, 1999.

[2] Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, Nov 1995.

[3] Gavin Lowe. Breaking and fixing the Needham-Schoeder public-key protocol using FDR. *Lecture Notes in Computer Science*, (1005):147–166, 1996.

[4] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), Dec 1978.

[5] Makoto Tatebayashi, Natsume Matsuzaki, and David B. Newman, Jr. Key distribution protocol for digital mobile communication systems. In Giles Brassard, editor, *Advances in Cryptology – CRYPTO '89*, number 435 in Lecture Notes in Computer Science, pages 324–334, New York, 1990. Springer-Verlag.