Lost in Just the Translation

Translation-based Steganography without Cover Source Transmission

Ryan Stutsman,
Mikhail Atallah,
Krista Grothoff
Purdue University

Christian Grothoff,
UCLA

Outline

- Background
  - Translation-based Steganography
    - Hide information in noise that occurs in natural language translation
  - Lost in Translation (LiT) Implemented System
    - A protocol that uses Translation-based Steganography to encode information
- Lost in Just the Translation (LiJtT)
  - A new, different protocol that overcomes limitations in the LiT protocol

Translation as a cover for hiding info

- Automated Natural Language Translation
  - Natural Language translation is a noisy process
  - Far from perfect
  - Availability of low-quality translations makes alterations plausible

An example from Babelfish

- The following German text was taken from a Linux Camp website:
  - “Keine Sorge, sie sind alle handzahm und beantworten auch bereitwillig Fragen rund um das Thema Linux und geben gerne einen kleinen Einblick in die Welt der Open-Source.”
- A reasonable English translation would be the following:
  - “Don’t worry, they are all tame and will also readily answer questions regarding the topic ‘Linux’ and gladly give a small glimpse into the world of Open Source.”
- Babelfish gave the following translation:
  - “A concern, it are not all handzahm and answer also readily questions approximately around the topic Linux and give gladly a small idea of the world of the open SOURCE.”

Lost in Translation (LiT)

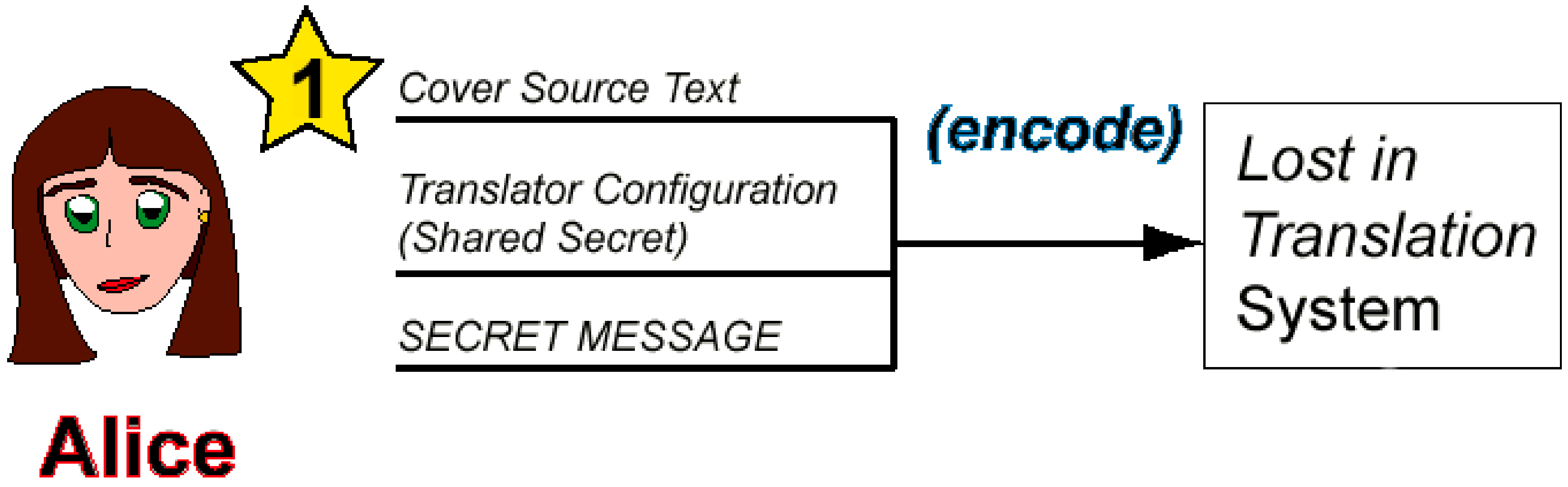
The Protocol:

We assume Alice and Bob have a shared secret in advance – in this case, it is the translation-system configuration (training corpus, etc).

To send a message: First, Alice chooses a source text – possibly public. It does *not* have to be secret.

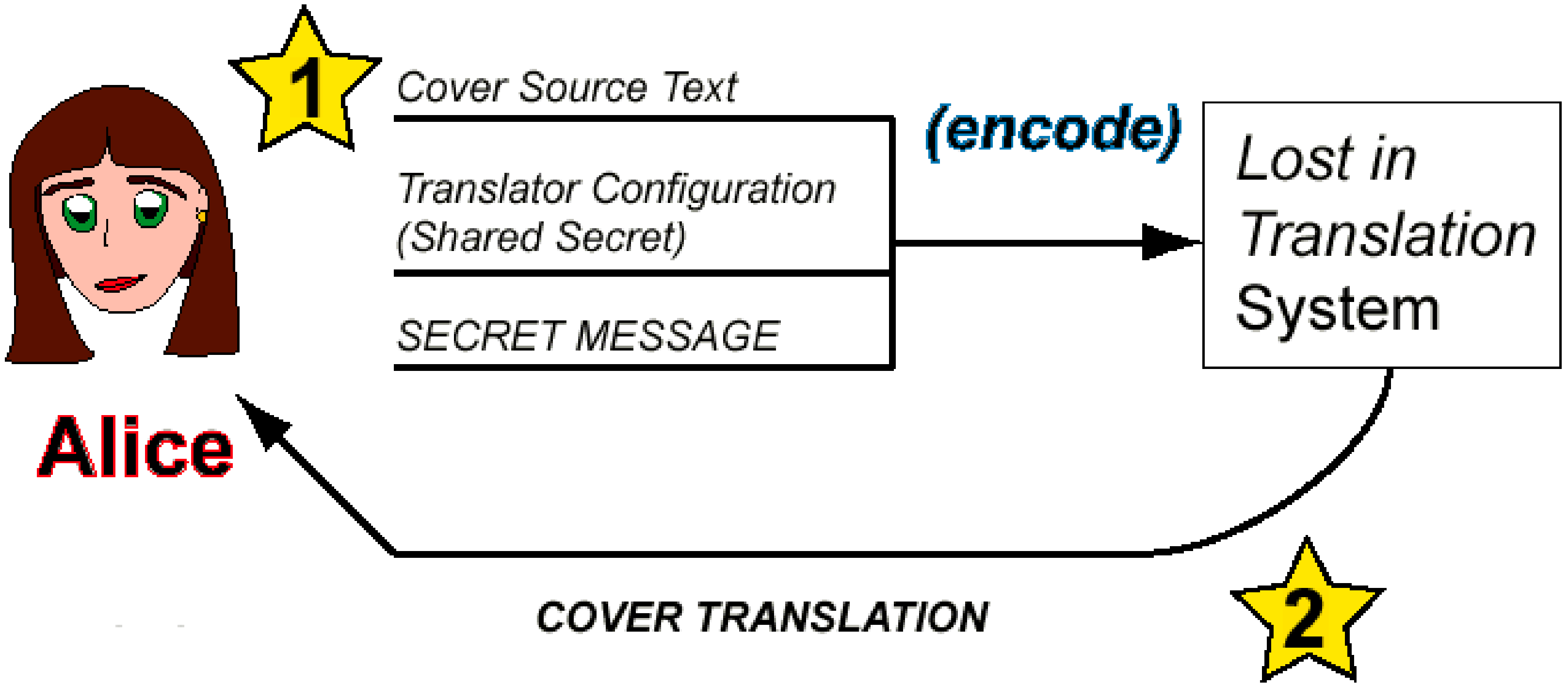
LiT Protocol: Step 1

- Alice feeds the shared-secret configuration, her cover source text, and the secret message into the LiT system.


1
Cover Source Text
Translator Configuration (Shared Secret)
SECRET MESSAGE
(encode)
Lost in Translation System
Alice

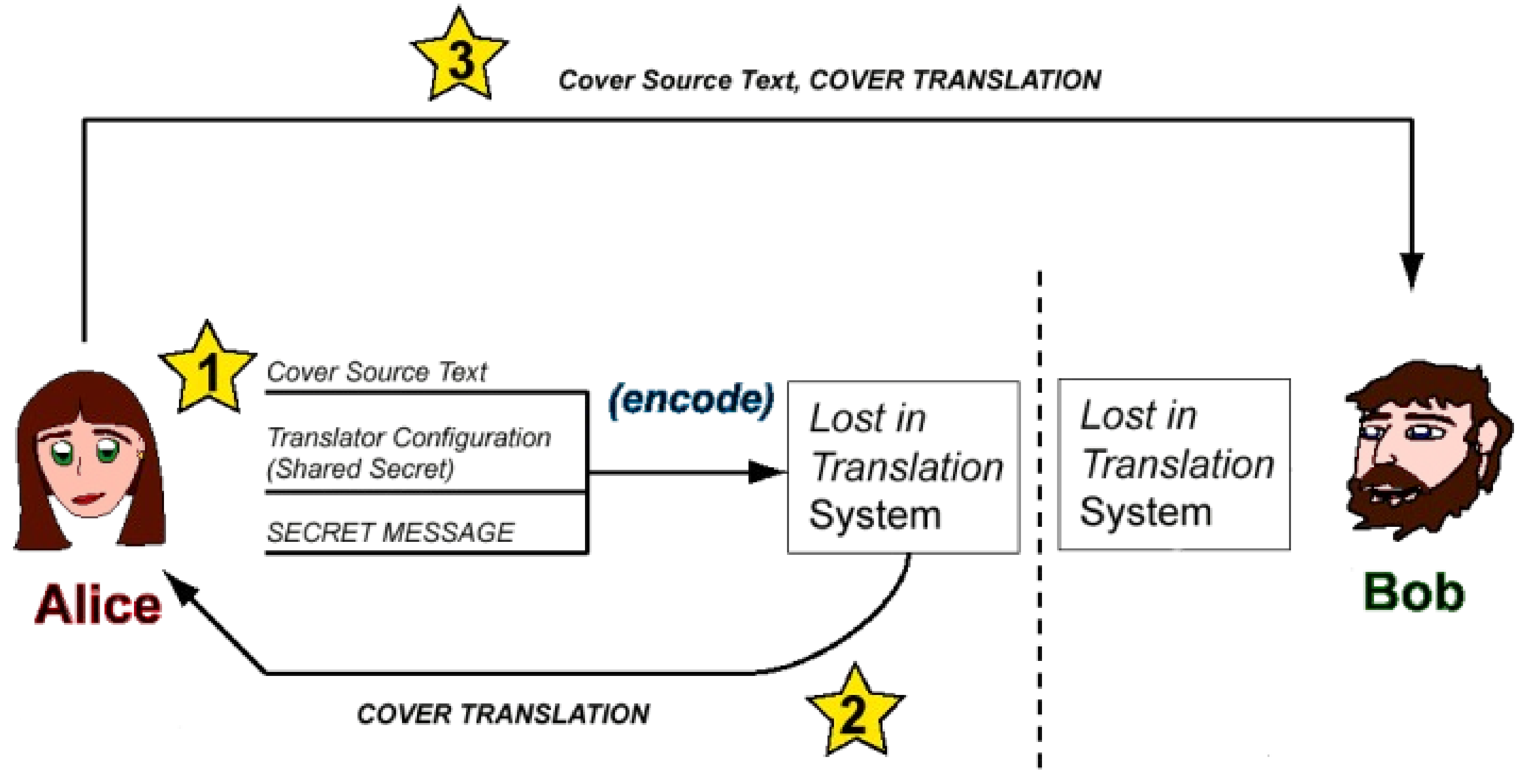
LiT Protocol, Step 2

- System encodes the secret message within a cover translation.


1
Cover Source Text
Translator Configuration (Shared Secret)
SECRET MESSAGE
(encode)
Lost in Translation System
Alice
COVER TRANSLATION
2

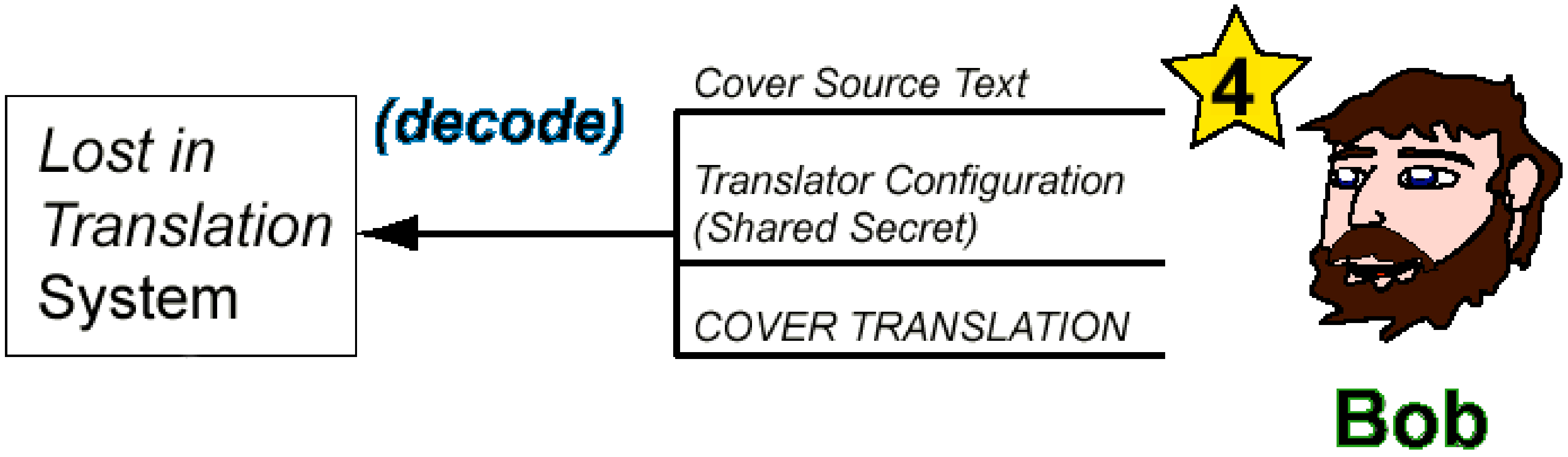
LiT Protocol: Step 3

- Alice sends the cover source text (or a reference to it) and the cover translation to Bob


3
Cover Source Text, COVER TRANSLATION
1
Cover Source Text
Translator Configuration (Shared Secret)
SECRET MESSAGE
(encode)
Lost in Translation System
Lost in Translation System
Alice
Bob
COVER TRANSLATION
2

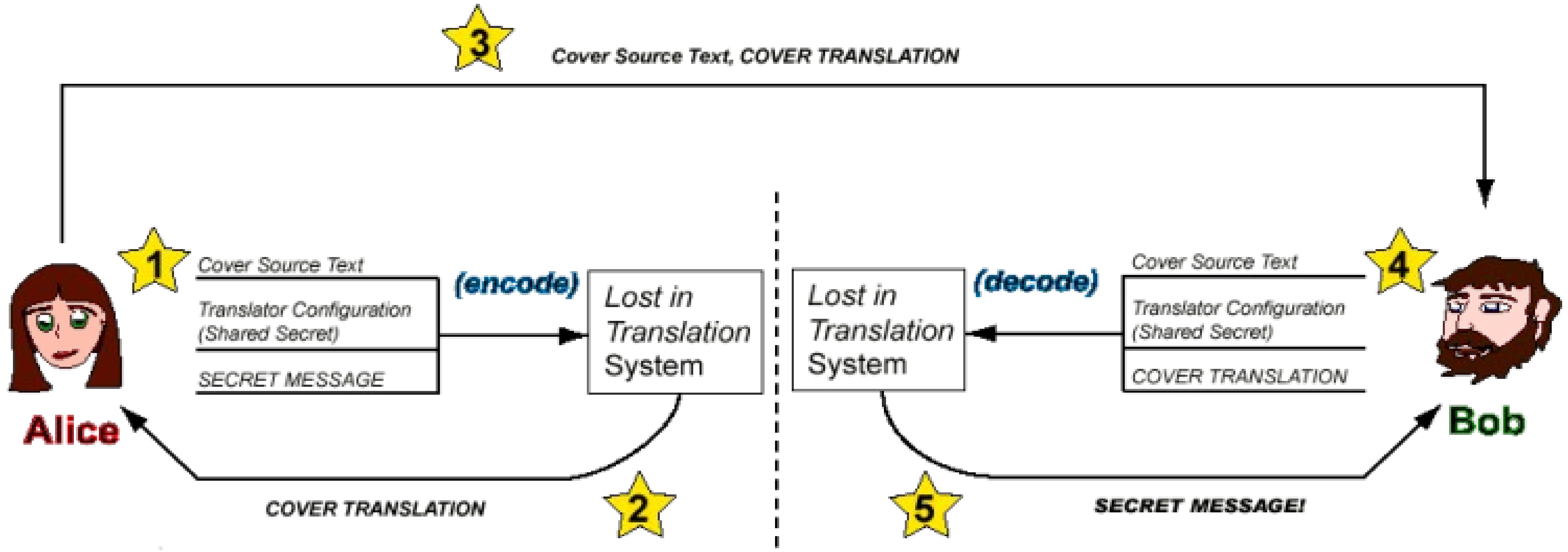
LiT Protocol: Step 4

- Bob, who already has the system configuration, feeds this, the cover source text and cover translation into his LiT system.

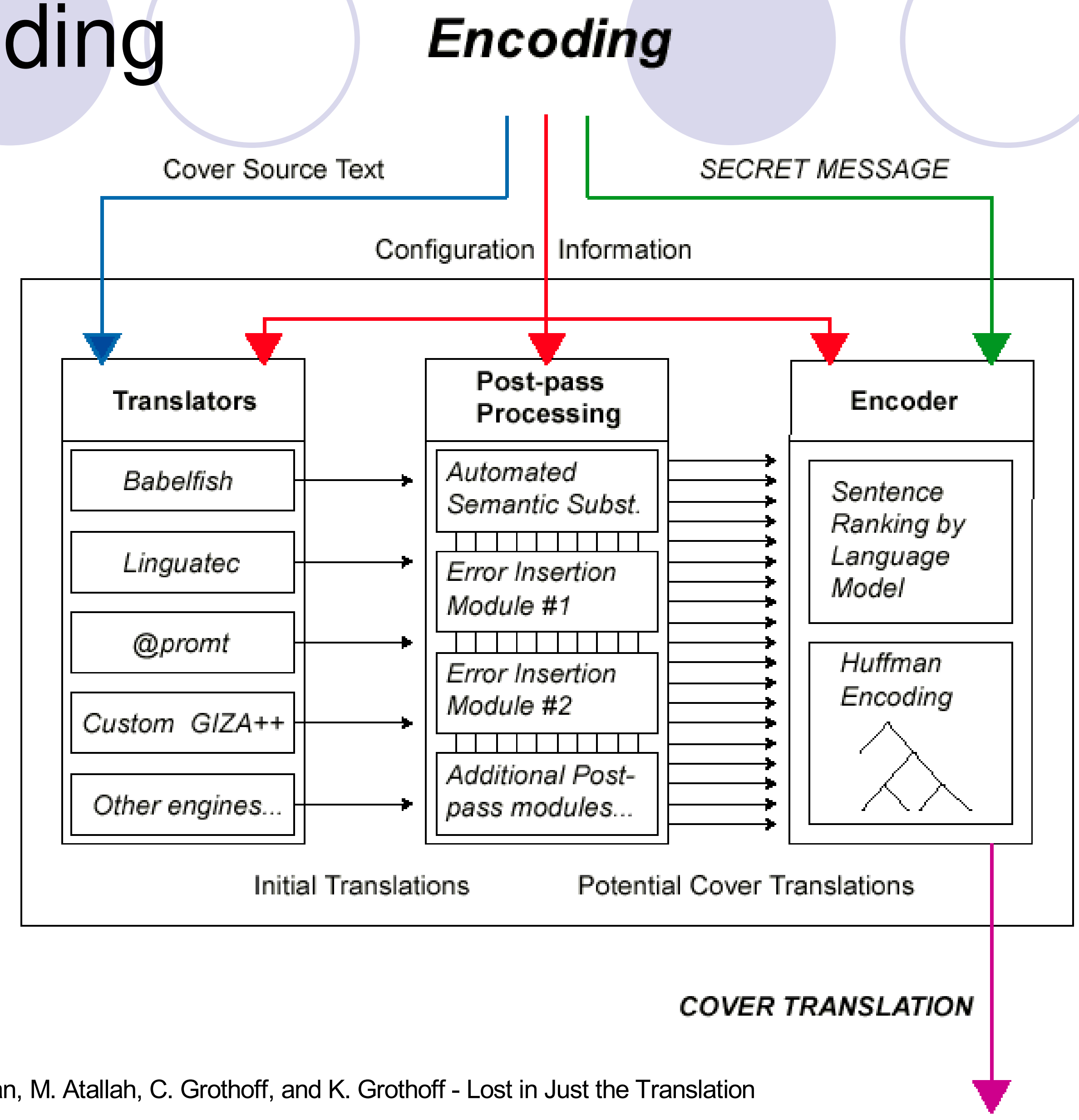

Lost in Translation System
(decode)
Cover Source Text
Translator Configuration (Shared Secret)
COVER TRANSLATION
4
Bob

LiT Protocol: Step 5

- The system matches the cover translation with the appropriate translation values and gives Bob the secret message.

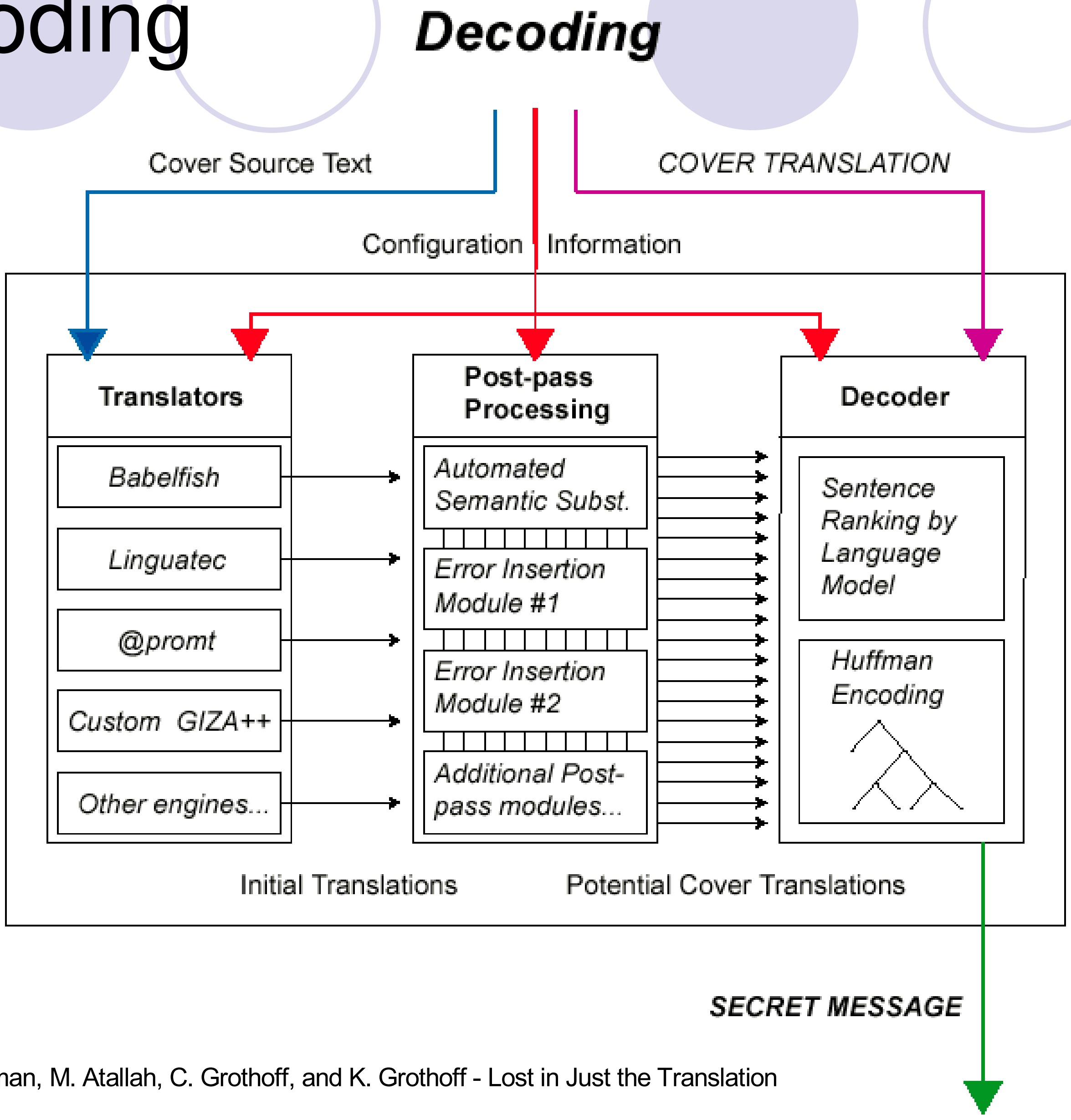

3
Cover Source Text, COVER TRANSLATION
1
Cover Source Text
Translator Configuration (Shared Secret)
SECRET MESSAGE
(encode)
Lost in Translation System
Alice
COVER TRANSLATION
2
Lost in Translation System
(decode)
Cover Source Text
Translator Configuration (Shared Secret)
COVER TRANSLATION
4
Bob
5
SECRET MESSAGE!

Encoding



Encoding
Cover Source Text
SECRET MESSAGE
Configuration Information
Translators
Babelfish
Linguatec
@promt
Custom GIZA++
Other engines...
Post-pass Processing
Automated Semantic Subst.
Error Insertion Module #1
Error Insertion Module #2
Additional Post-pass modules...
Encoder
Sentence Ranking by Language Model
Huffman Encoding
Initial Translations
Potential Cover Translations
COVER TRANSLATION

Decoding


Decoding
Cover Source Text
COVER TRANSLATION
Configuration
Information
Translators
Babelfish
Linguatec
@promt
Custom GIZA++
Other engines...
Post-pass Processing
Automated Semantic Subst.
Error Insertion Module #1
Error Insertion Module #2
Additional Post-pass modules...
Decoder
Sentence Ranking by Language Model
Huffman Encoding
Initial Translations
Potential Cover Translations
SECRET MESSAGE

Advantages

- LiT hides within the limits of machine translation – as MT models change, so can our system
- Avoids "generation" problem by trying to mimic the results of an imperfect transformation, *not* by mimicking correct, human-produced text
- Secret key (implementation, training corpora and configuration) allows for many possible encoders
- Cover text can be public and obtained from public sources

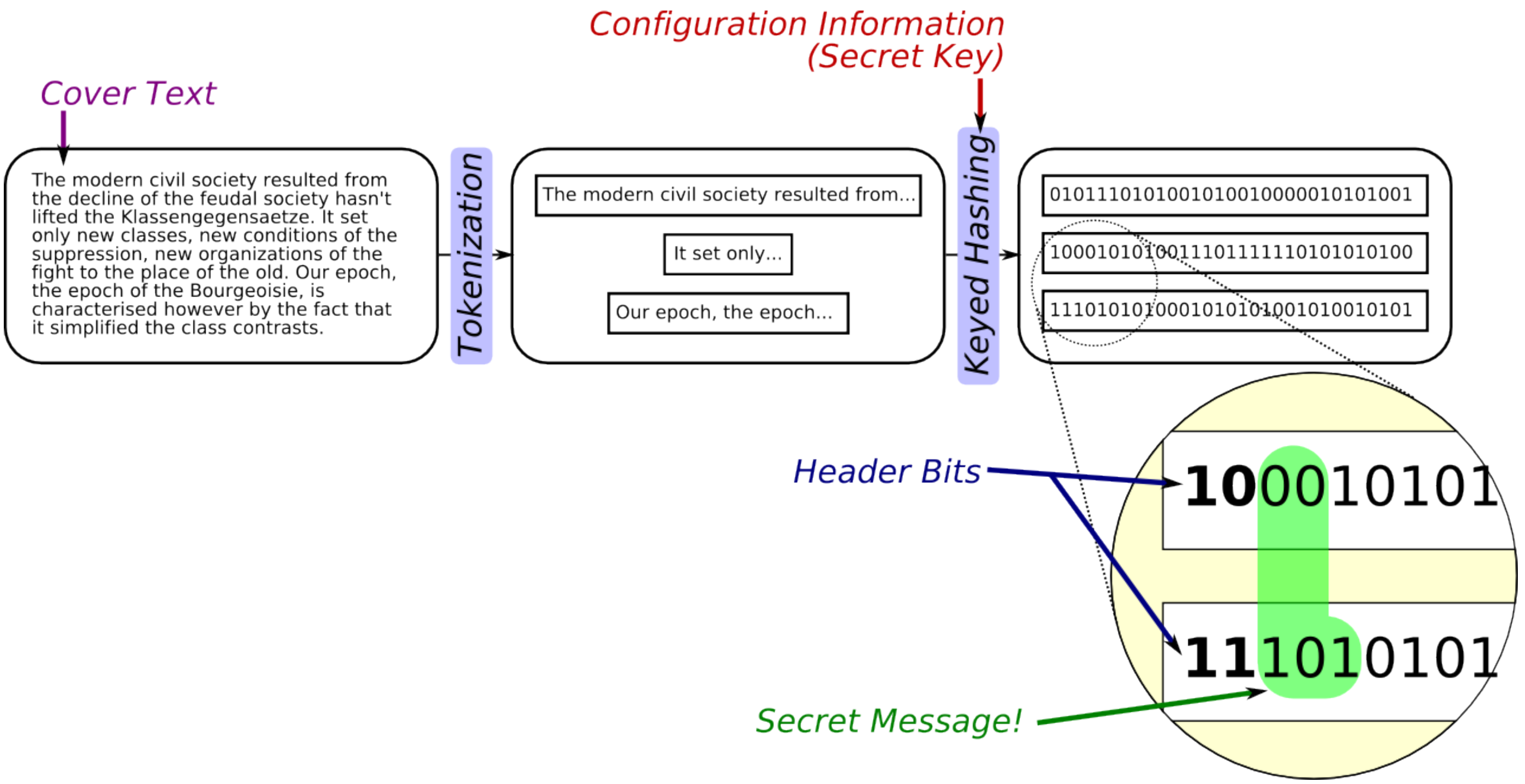
Disadvantages

- Attacker could spot obvious inconsistencies
  - Same sentence translated in two ways
  - Certain mistakes made inconsistently (“feet”/“foots”)
- Need to transmit both source text (or a reference to it) and translation
  - Steganographers are generally uncomfortable with transmitting the original source object
- Both of these issues are addressed
  - A new protocol – Lost in Just the Translation

Lost in Just the Translation (LiJtT)

- Transmission without the source text
- Encode data based on keyed hashes of translations
- More complicated encoder
- Straightforward decoder
    - A good place to begin to get an overview of the protocol

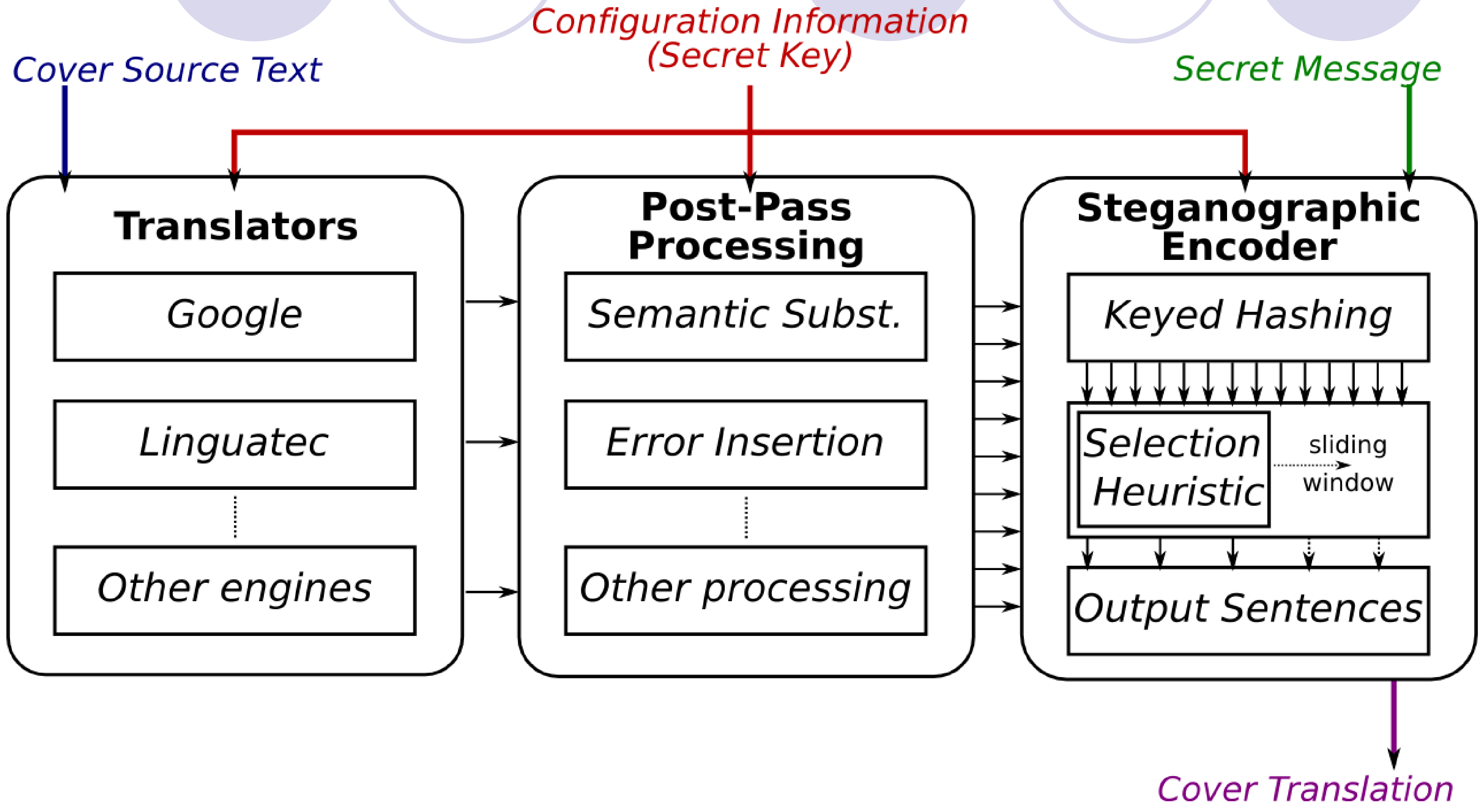
Decoding


Cover Text
Configuration Information (Secret Key)
The modern civil society resulted from the decline of the feudal society hasn't lifted the Klassengegensaetze. It set only new classes, new conditions of the suppression, new organizations of the fight to the place of the old. Our epoch, the epoch of the Bourgeoisie, is characterised however by the fact that it simplified the class contrasts.
Tokenization
The modern civil society resulted from...
It set only...
Our epoch, the epoch...
Keyed Hashing
01011101010010100100000010101001
10001010100111011111101010100
11101010100010101010010100101 01
Header Bits
100010101
111010101
Secret Message!

Decoding embedded messages

- Tokenize according to original LiT decoder
  - LiJtT generally works best with sentences as tokens
- Compute keyed hash using shared secret k
- Split the bits of the hash into two pieces
  - First h bits encode an integer b
  - Bits $[h + 1, h + b]$ contain the hidden information

Encoding



Configuration Information (Secret Key)
Cover Source Text
Secret Message
Translators
Google
Linguatec
Other engines
Post-Pass Processing
Semantic Subst.
Error Insertion
Other processing
Steganographic Encoder
Keyed Hashing
Selection Heuristic
sliding window
Output Sentences
Cover Translation

Which Translation to Select

- Hashes of translations scored and ranked
  - Metric of how well a hash matches what we are trying to encode
- Select longest matching sequence
  - Maximizes carrying capacity for that sentence
  - Smarter selection: Adjust for errors
    - Select largest *b* minus the cost of correcting errors
- Error correction is used in case of inability to encode

Encoding Messages

- Sender/Receiver share secret key
- *h*, the number of header bits, must be agreed upon
  - Can be shared or transmitted in first hash, for example
- Sender selects a cover source text
  - Can be public or secret (more secure)
    - Multiple translations are plausible
    - Source is not necessary after encoding; can be discarded

Choosing *h*

- *h* has a strong impact on capacity
  - Too small: little data can be stored per sentence
  - Too large: lowers the probability of a hash matching the bits $[h + 1, h + b]$
- $1 \leq h \leq 4$ have been most effective in our testing

Issues Processing a Sliding Window

- Maximize encoding, *but*
- Avoid damage to later windows
  - Best storage for a sentence may hinder later storage
- A heuristic approach
  - Exhaustive search and dynamic programming are too expensive

Tracking/Scoring State of Computation

- Tentative choices made; can backtrack
- Track the overall storage of a set of selections up until the current sentence using a tuple containing
    - Translations, total number of bits and errors encoded up to current point
- Maintain a pool of *t* tentative choices at any point throughout process (*t* is configurable)

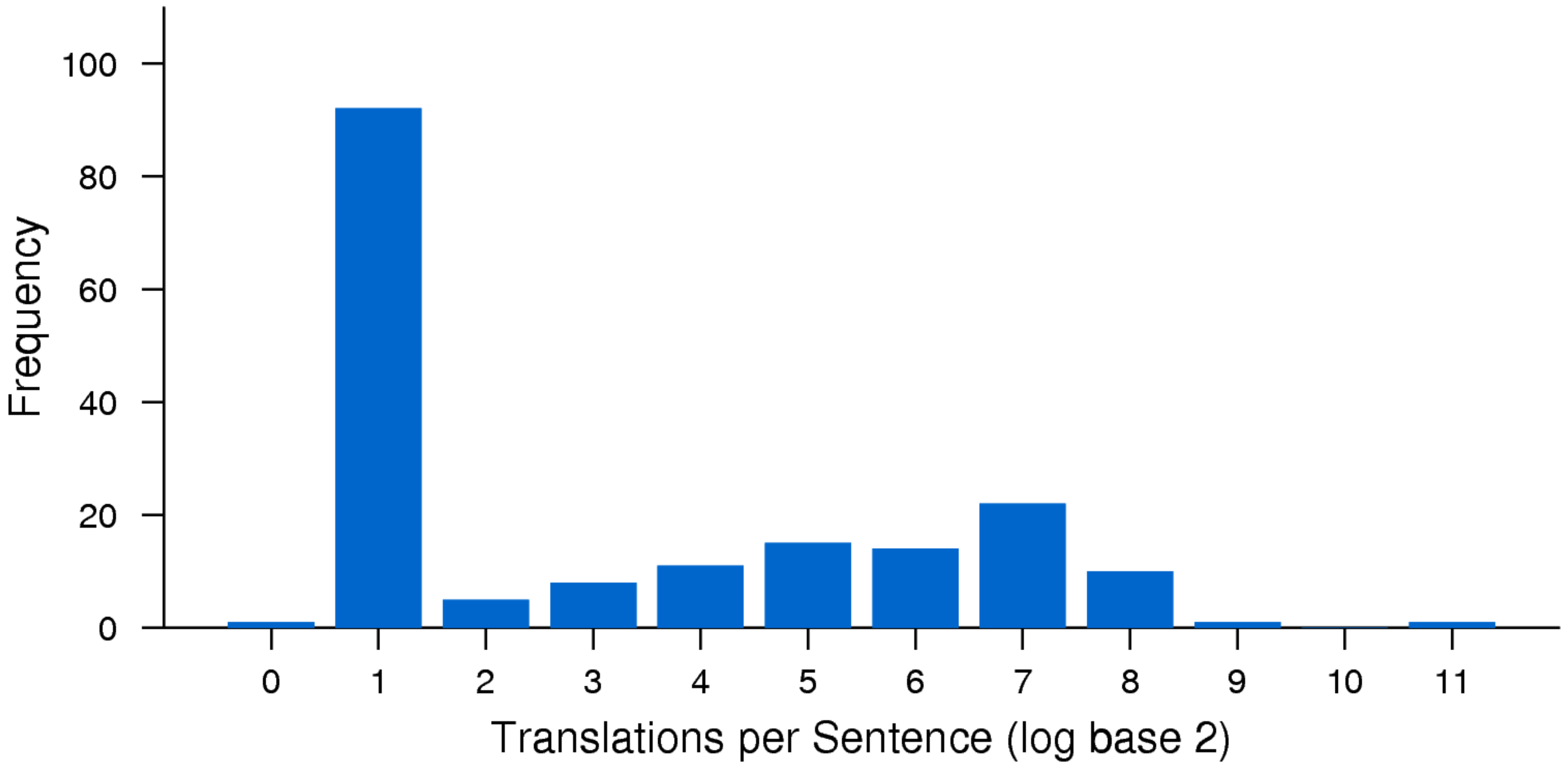
Tracking/Scoring (cont.)

- For each tentative choice a new set of sub-choices is computed
- The new set of tentatives (larger than *t*) is quality-sorted and the best *t* selected

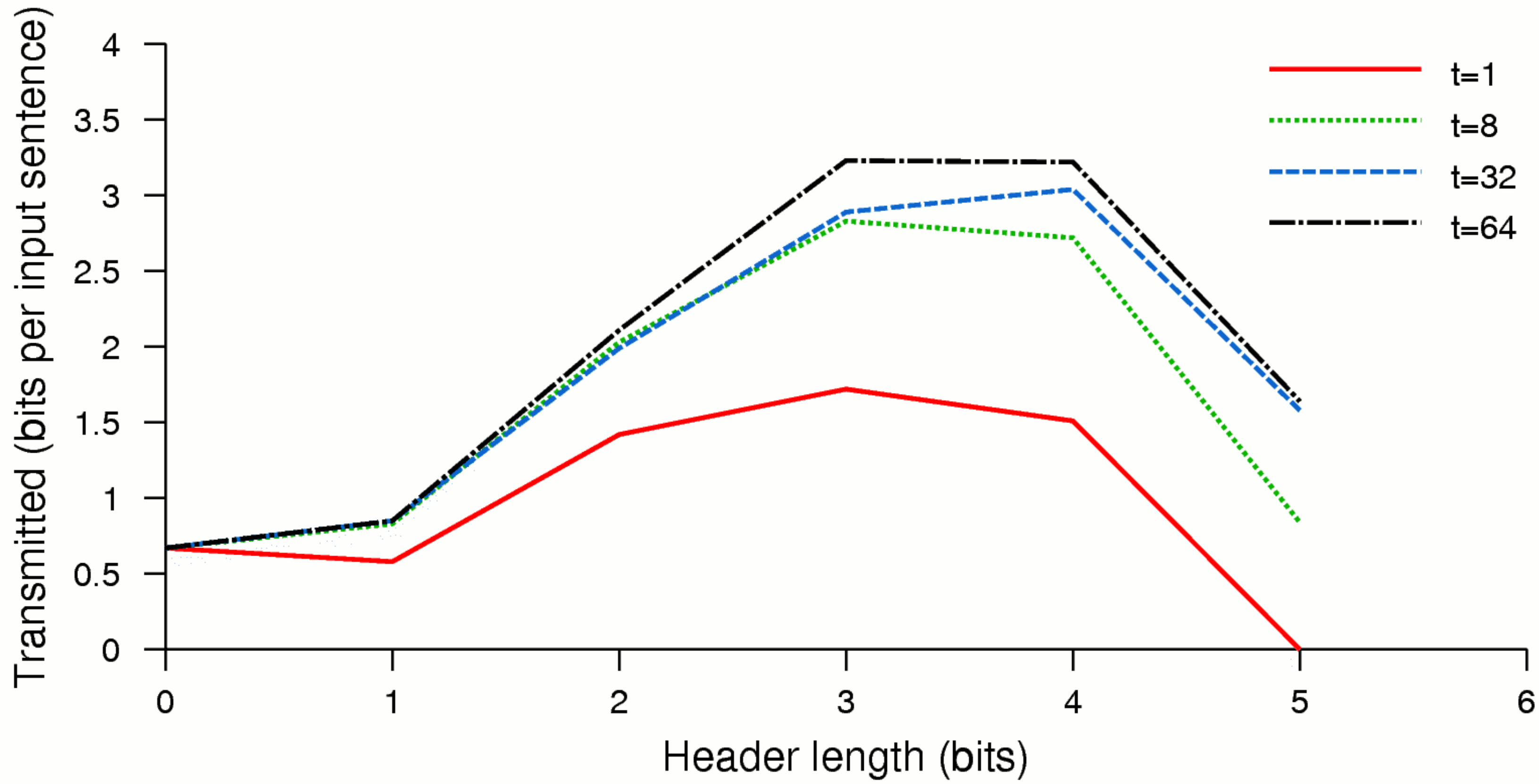
Experimental Results

- Initial implementation built on LiT infrastructure
- Employs various publicly accessible MT systems
  - Initial tests only include two public MT systems

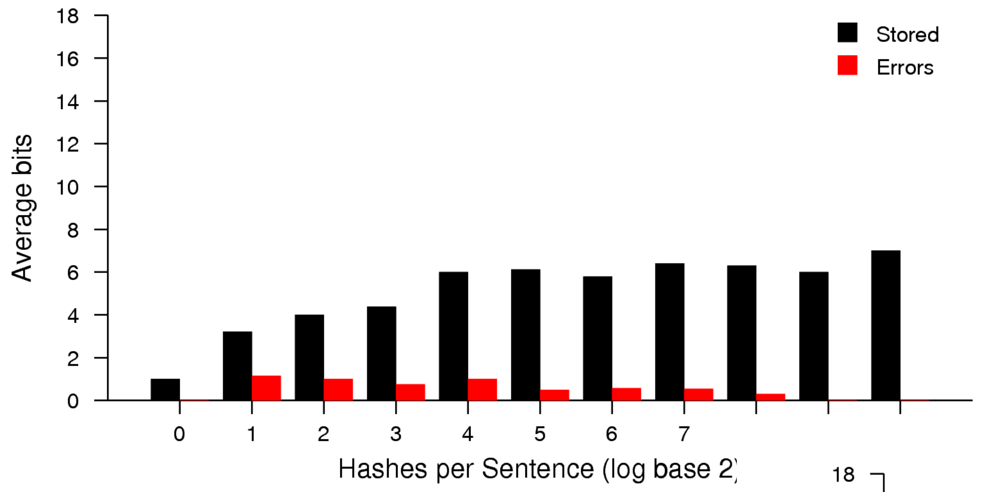
Translation Count Distribution



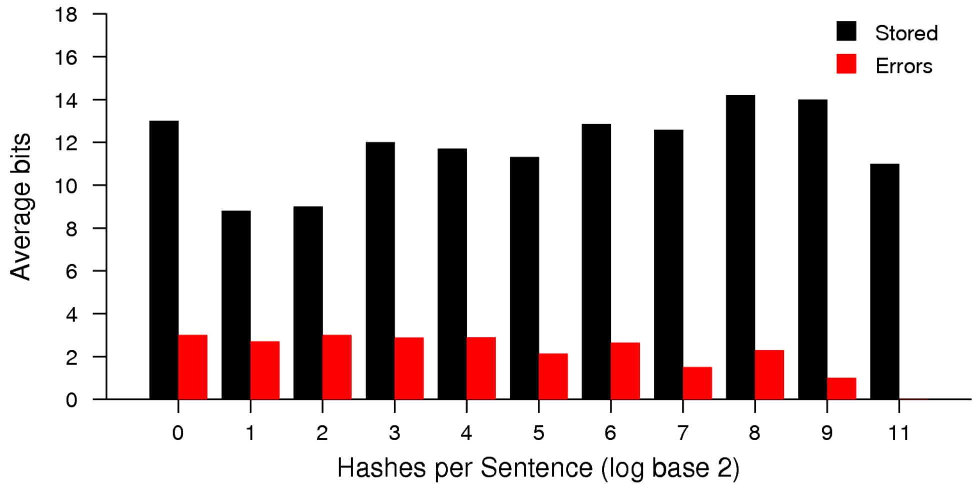
Effect of *h* and *t*



Data Rate Variance



Storage/Errors when choices threshold is 1



Storage/Errors when choices threshold is 32

Information Density

- Original LiT Implementation
  - 0.37%
- LiJtT Implementation
  - 0.28% ~581 bits for 25.6 KB of transmitted text
- The initial implementation transmits about 25-50% less data in the same transmission as the original protocol
- LiJtT with 1 additional Human Translation
  - 0.33%

Conclusions

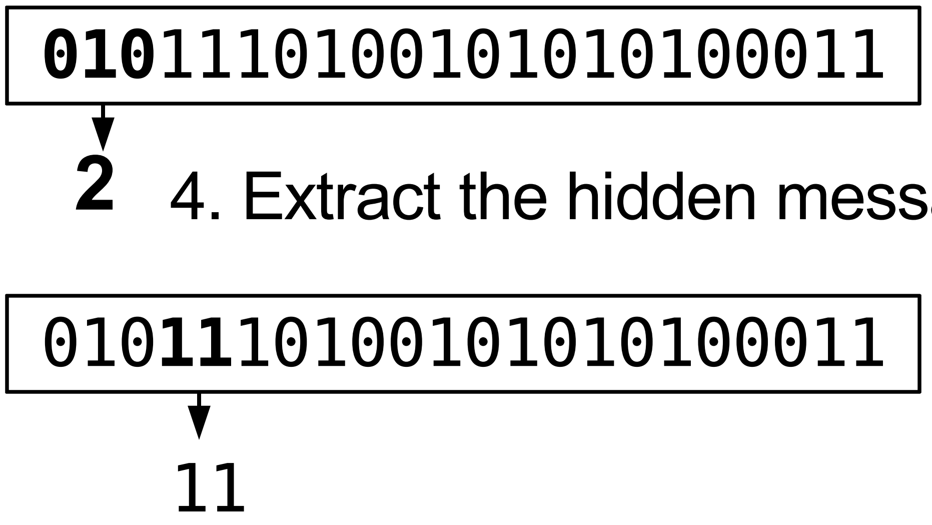
- Transmission of the hidden message can be done without requiring the original source text
- Protocols can be adapted to thwart adversarial analysis by modifying the language model in the encoder

The Implementation!

- LiJtT uses PHP and MySQL and is available under the GNU General Public License
- Prototype is available at http://www.cs.purdue.edu/homes/rstutsma/stego/
- The software itself is available for download via subversion at https://gnunet.org/projects/stego/

Example Decoding from a Single Hash

3. Interpret header (*h*) bits (assume *h* = 3) as an integer *b*

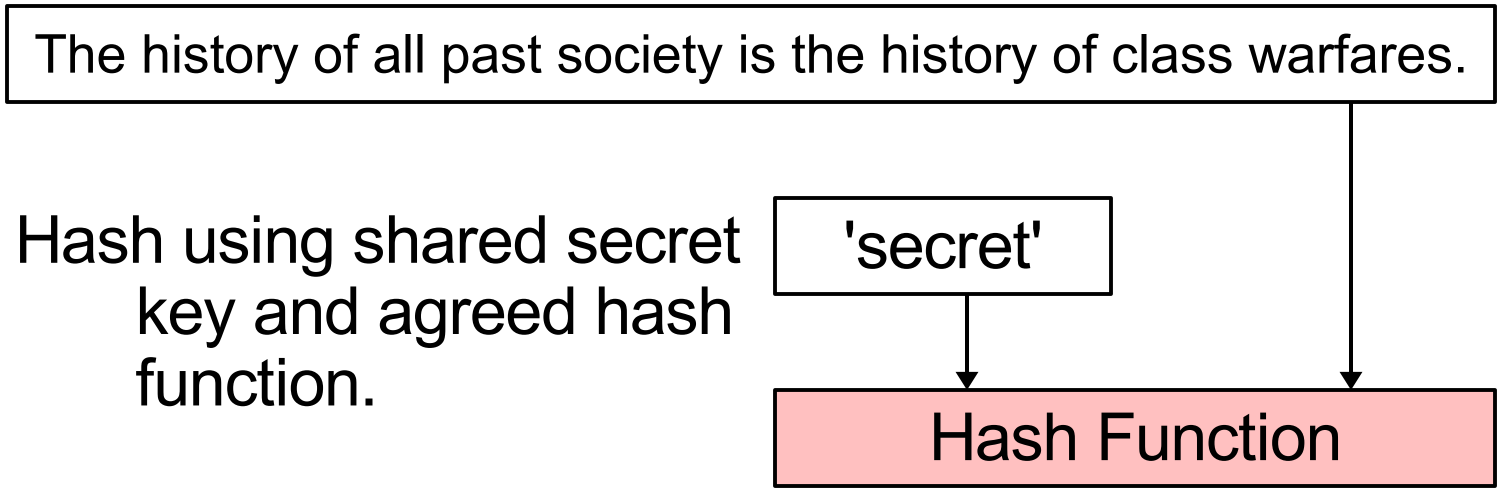


4. Extract the hidden message in bits [*h* + 1, *h* + *b*]

5. Concatenate extracted bits to message extracted from previous tokens and repeat for next token.

Example Decoding from a Single Hash

1. Get next token from the transmitted text (usually by sentence or phrase depending on configuration).


The history of all past society is the history of class warfares.
'secret'
Hash Function

2. Hash using shared secret key and agreed hash function.

Resulting Bitstream

```
0101111010010101010100011
```

Hash Selection Example

- A walkthrough
  - Hash ranking/selection to find the set of sentences that best encodes the secret message bit sequence
- Parameters we need to know
  - Error correction cost can be estimated to be 3 bits for every error (we treat all errors as substitution errors)
  - Let the number of header bits (*h*) per hash be 2

Hash Selection Example

h = 2 (# of header bits per hash)
Estimated cost of making a substitution error is 3 bits

Bitstream to hide: `011001011`

Hashes for translations of the current input sentence:

1001...	Score: 2 bits stored – (3 * 0 errors) = 2 points ☆
11111..	Score: 3 bits stored – (3 * 1 errors) = 0 points

If this were the entire picture the first translation would be selected.

Hash Selection Example

We move on to the next sentence with “01” already hidden.

Bitstream to hide: `~~01~~1001011`

Hashes for translations of the current input sentence:

$\boxed{\texttt{11011..}}$ Score: 3 bits stored – (3 * 3 errors) = -6 points

$\boxed{\texttt{11001..}}$ Score: 3 bits stored – (3 * 2 errors) = -3 points ☆

Both are bad choices so choose the one that causes the least damage.

Total score is -1 adding the score from the previous sentence.
Could we have done better? Let's start over and try again.

Smarter Hash Selection Example

Bitstream to hide: `011001011`

Hashes for translations of the current input sentence:

1001...	Score: 2 bits stored – (3 * 0 errors) = 2 points
111**1**1..	Score: 3 bits stored – (3 * 1 errors) = 0 points ☆

We consider the error to be a substitution error; even though we encoded an error we still consider the bit hidden and move on.

Where does this leave us for the next sentence?

Smarter Hash Selection Example

We move on to the next sentence with "011" already hidden.

Bitstream to hide: `~~011~~001011`

Hashes for translations of the current input sentence:

$\boxed{\texttt{11011..}}$ Score: 3 bits stored – (3 * 1 errors) = 0 points

$\boxed{\texttt{11001..}}$ Score: 3 bits stored – (3 * 0 errors) = 3 points ☆

Despite substitution error on the first selection we ended up better off.

Total storage using this route: 3 points