Information Theory and Security: Quantitative Information Flow

Pasquale Malacaria

pm@dcs.qmul.ac.uk

School of Electronic Engineering and Computer Science

Queen Mary University of London

Plan

Give some answers to the following questions:

1. Why Information Theory?
2. What is leakage of confidential data?
3. How to measure leakage?
4. How to reason about leakage?
5. How to implement a leakage analysis?

From horses to the Linux Kernel

The Problem

Consider the following simple program

```
if (password==guess) access=1; else
access=0;
```

unavoidable leakage of confidential information:

1. Observing `access=1`: guessed the right password
2. Observing `access=0`: eliminated one possibility from the search space.
3. So the real security question is not whether or not programs leak, but how much.
4. Some QIFfers: Chatzikokolakis, Chotia, Clark, Chen, Heusser, Hunt, Kopf, Malacaria, McCaimant, Mu, Palamidessi, Panangaden, Rybalchenko, Smith, Tereauchi.

Why Information Theory?

Shannon's entropy measures the **information content** of a random variable.
Consider a 4 horses race: the random variable W means "the winner is".
W can take four values, value i standing for "the winner is the $i-$th horse".

Information content of a random variable = *the minimum space needed to store and transmit the possible outcomes of a random variable.*

Some intuitions on Information Theory

Shannon's entropy measures *the minimum space needed to store and transmit the possible outcomes of a random variable.*

1. If we know who will win (probability 1), then no space needed to store or transmit the information content of W, i.e. W has 0 information content.
2. Other extreme: all 4 horses are equally likely to win. Then the information content of W is 2 because using 2 bits is possible to store 4 values.
3. If there were only two possible values and they were equally likely then the information content of W would be 1 because in 1 bit is possible to store 2 values.

Some intuitions on Information Theory

Hence entropy of W, $H(W)$ should take values $0, 2, 1$ respectively when W follows the distributions

1. $p_1 = 0, 0, 0, 1$ (for the first case),
2. $p_2 = 1/4, 1/4, 1/4, 1/4$ (for the second case) and
3. $p_3 = 1/2, 1/2, 0, 0$ (for the third case).

Use Shannon's entropy formula

$$H(W) = -\sum_i p_i \log_2 p_i$$

e.g.

$$H(p_2) = -\sum_i 1/4 \log_2 1/4 = 4 * (1/4 \log_2(4)) = 2$$

Information=Uncertainty

1. If we know who will win (probability 1) then uncertainty on (the value of) W = 0.
2. Other extreme: all 4 horses are equally likely to win. Then uncertainty on W (wrt 4 possibilities) is maximal = 2 bits (4 possible values).
3. If there were only two possible values and they were equally likely then the information content of W = 1 bit (2 possible values).

$H(W) =$ Information content of W= Uncertainty about W

Some intuitions on Information Theory

Related notions: **Conditional Entropy:** what is the uncertainty on W given knowledge of the horse arriving last?

- If we know the winner then knowing the loser won't change the uncertainty on the winner
- If all 4 horses equally likely to win then the loser will eliminate one possible winner
- If 2 out of 4 horses are possible winners then the loser will not affect the uncertainty about the winner (assuming the last is not one of the two possible winners)

$$H(W|\text{ Last }) = 0, \log_2(3), \log_2(2) \text{ respectively}$$

Some intuitions on Information Theory

Conditional Entropy: what is the uncertainty on W given knowledge of the horse arriving last?
Easy formal definition:

$$H(X|Y) = H(X,Y) - H(Y)$$

$H(X,Y)$ is the joint entropy of X and Y and is just the entropy defined on the joint probabilities:

$$H(X,Y) = \sum_{x,y} p(x,y) \log_2 p(x,y)$$

$H(X|Y)$ =Uncertainty about X,Y minus uncertainty on Y

Some intuitions on Information Theory

$$H(X|Y) = H(X,Y) - H(Y)$$

$H(W|$ Last $) = 0, \log_2(3), \log_2(2)$ respectively

Some intuitions on Information Theory

Related notions:

- **Mutual Information:** difference in uncertainty on W before and after knowledge of the horse arriving last?

$$I(W;\ \text{Last}\ ) = H(W) - H(W|\ \text{Last}\ ) = 0,2 - \log_2(3), 1 - \log_2(2) = 0$$

What is Leakage?

Leakage=**difference** in the **uncertainty** about the secret h **before and after** observations O on the system:

$$H(h) - H(h|O) = I(h;O) \text{ (mutual information)}$$

- In general we also want to take into account contextual information
- **Leakage: Conditional Mutual information:** $I(h;O|L)$ **difference** in the **uncertainty** about the secret h **before and after** observations on the system O given contextual information L
- the correlation between secret h and observations O given L, a *measure of the information* h, O *share given* L

What is Leakage?

Leakage=**difference** in the **uncertainty** about the secret h **before and after** observations O on the system:

- **Leakage: Conditional Mutual information:** $I(h; O|L)$ **difference** in the **uncertainty** about the secret h **before and after** observations on the system O given contextual information L
- This definition can be used for leakage in programs and probabilistic systems or loss of anonymity in Anonymity protocols ((Chastikokolakis-Palamidessi-Panangaden, Chen-Malacaria)

Channel Capacity

- Leakage=**difference** in the **uncertainty** about the secret h **before and after** observations O on the system:
- Question: what is the maximum leakage for a system?
- Consider all possible distribution on the secret and pick the maximum leakage in this set
-

$$CC = \max_{h} I(h; O|L)$$

Some intuitions on Information Theory

If we consider leakage in deterministic programs things simplify; in fact:

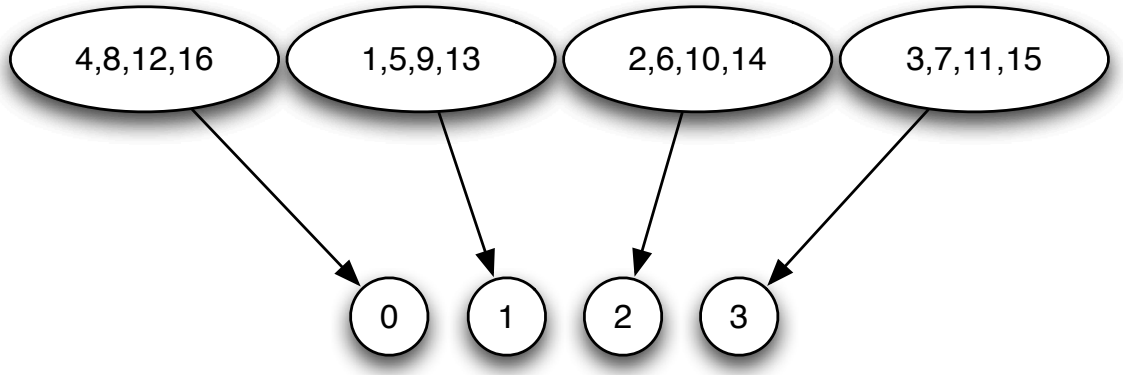
$$I(h; O|L) = H(O|L) - H(O|h, L)$$

a program is a function from inputs to output $P(h, L) = O$, so

$$H(O|h, L) = 0$$

Example

Assume `h` is 4 bit ($1 \ldots 16$).
`P(h)` is the program `l = h % 4;`



$$H(O) = -\sum p \log_2(p) = 4 \, \frac{1}{4} \log_2(4) = 2 \text{ bit}$$

Meaning: on average observing one output will leave you with a 2 bits (four values) uncertainty about the secret
Notice the preimage of `P(H)` (i.e. O^{-1}) which *partitions* the high inputs.

Partitions vs Random Variables

We can see partitions over a space equipped with a probability distribution as a random variable.
Usually a random variable is defined a map f from a space equipped with a probability distribution to a measurable space.
So f^{-1} is a partition on a space equipped with a probability distribution

The Lattice of Information

Leakage=$H(O)$ where O is the random variable "**output observations**" of the program.
It corresponds to the partition on the high inputs given by O^{-1}.
observation = partial information = sets of indistinguishable items

LoI and Information Theory

Apparently LoI and Information theory have nothing in common.
A surprising result by Nakamura shows otherwise:

> **Theorem (Nakamura):** If LoI is built over a probabilistic space then the *best measure* is Shannon Entropy

Measure here is a lattice semivaluation, i.e. a real valued map ν s.t.

$$\nu(X \sqcup Y) \leq \nu(X) + \nu(Y) - \nu(X \sqcap Y) \quad (1)$$

$$X \sqsubseteq Y \text{ implies } \nu(X) \leq \nu(Y) \quad (2)$$

(No stronger notion is definable on LoI)

LoI and Information Theory

Shannon point: Information Theory measures the amount of information. It doesn't describe what the information is about.
E.g. a coin toss and the US presidential race: both described by $H(X) \leq 1$ So what does describe information?

> Answer: A set of processes that can be translated between each other without losing information

$$d(X,Y) = H(X|Y) + H(Y|X)$$

A set of processes s.t. for all X, Y, $d(X,Y) = 0$
d defines a pseudometric on a space of random vars, i.e. a metric on the information items.

LoI and Information Theory

Shannon point 2: define the following order on this space:

$$X \geq_d Y \Leftrightarrow H(Y|X) = 0$$

The intuition here is that X provides complete information about Y, or equivalently Y has less information than X, so Y is an abstraction of X (some information is forgotten).

$$X \sqsubseteq Y \Leftrightarrow X \leq_d Y$$

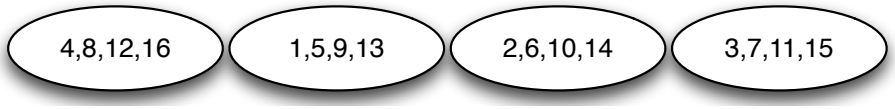
So LoI is also the lattice of information in Shannon's sense

Quantifying Leakage and Partitions

Leakage: uncertainty about the inputs after observing the outputs of a program

Measured using Shannon Entropy using the following steps

1. Take some code `l = h % 4`
2. Interpret the code in LoI: find partition on high inputs



3. Quantify using Entropy (Measure the partition)

$$-\sum p \log_2(p)$$

How to reason about leakage?

We give an example how to reason about loops (Malacaria POPL 2007):
Consider

```
l=0;
while(l < h) {
if (h==2) l=3 else l++
}
```

	It=0	It=1	It=2	It=3
O	0	1,3	ϵ	3
h	0	1,2	ϵ	3

How to reason about leakage?

We can also use the Lattice of Information:

```
l=0;
while(l < h) {
if (h==2) l=3 else l++
}
```

	lt=0	lt=1	lt=2	lt=3
O	0	1,3	ϵ	3
h	0	1,2	ϵ	3

Implementing the analysis

Joint work with Jonathan Heusser
Similar ideas also in Backes, Kopf, Rybalchenko

Where we aim to be

```
static int
auth1_process_rhosts_rsa(Authctxt *authctxt, char *info, size_t infolen)
{
        int keybits, authenticated = 0;
        u_int bits;
        Key *client_host_key;
        u_int ulen;

        /*
         * Get client user name.  Note that we just have to
         * trust the client; root on the client machine can
         * claim to be any user.
         */
        client_user = packet_get_string(&ulen);

        /* Get the client host key. */
        client_host_key = key_new(KEY_RSA1);
        bits = packet_get_int();
        packet_get_bignum(client_host_key->rsa->e);
        packet_get_bignum(client_host_key->rsa->n);

        keybits = BN_num_bits(client_host_key->rsa->n);
        if (keybits < 0 || bits != (u_int)keybits) {
                verbose("Warning: keysize mismatch for client_host_key: "
                    "actual %d, announced %d",
                    BN_num_bits(client_host_key->rsa->n), bits);
        }
        packet_check_eom();
```

From Programs to Partitions

Given a partition and input probability distribution, quantification is simple. Just plug-in your measure.

More difficult is to get the partition for a program:

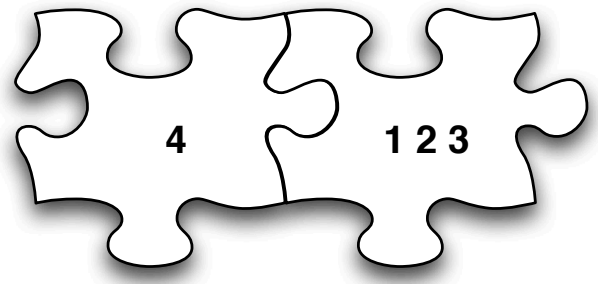
$$\Pi : Program \rightarrow Partition$$

Tool to calculate $\Pi(P)$ for subset of ANSI-C programs.

Automatically Calculating $\Pi(P)$

With 2 bit `pin`,

$$P \equiv \texttt{if(pin==4) ok else ko}$$



Partition defined by *number* and *sizes* of equivalence classes

Two step approach:

- Find a representative input for each possible output
- For each found input, count how many other inputs lead to the same output

Automatically Calculating $\Pi(P)$

Create two instances $P_{\neq}$ and $P_{=}$ out of P applying self-composition, inputs are h, h' and ouputs l, l'

$$\begin{aligned} P_{\neq}(i) &\equiv h = i; P; P'; \texttt{assert}(\texttt{l} \neq \texttt{l}') \\ P_{=}(i) &\equiv h = i; P; P'; \texttt{assert}(\texttt{l} = \texttt{l}') \end{aligned}$$

translated to SAT queries for SAT solving and model counting.

$P_{\neq}$ responsible for finding set of representative inputs S_{input} with unique outputs ($l \neq l'$)

$P_{=}$ model counts every element of S_{input}

Algorithm for $P_{\neq}$ by example

$$P \equiv \texttt{if(h==4) 0 else 1}$$

Input: $P_{\neq}$
Output: S_{input}
$S_{input} \leftarrow \emptyset$
$h \leftarrow random$
$S_{input} \leftarrow S_{input} \cup \{h\}$
while $P_{\neq}(h)$ *not unsat* **do**
  $(l, h') \leftarrow$ Run SAT solver on $P_{\neq}(h)$
  $S_{input} \leftarrow S_{input} \cup \{h'\}$
  $h \leftarrow h'$
  $P_{\neq} \leftarrow P_{\neq} \wedge l' \neq l$
end

$S_{input} = \{0, 4\}$ thus P has two equivalence classes

S_{input} is input to the algorithm for $P_{=}$

Algorithm for $P_=$ by example

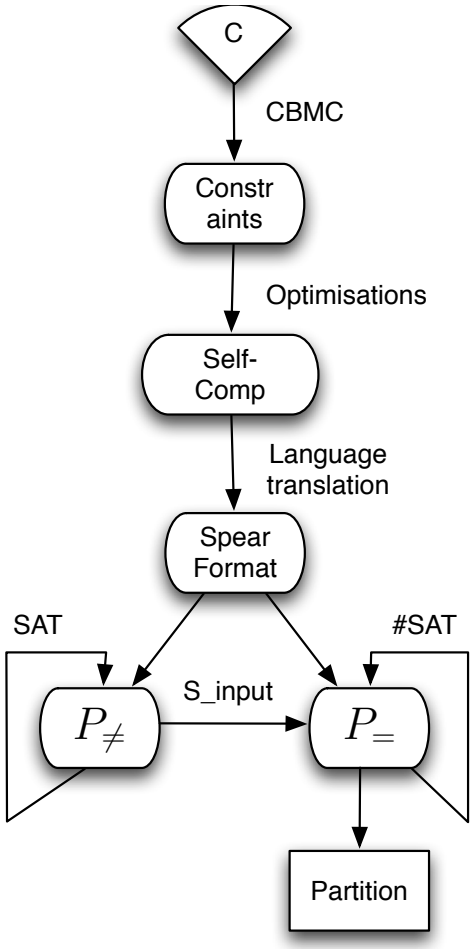
$$P \equiv \texttt{if(h==4) 0 else 1}$$

$$S_{input} = \{0, 4\}$$

Input: $P_=$, S_{input}
Output: M
$M = \emptyset$
while $S_{input} \neq \emptyset$ **do**
  $h \leftarrow s \in S_{input}$
  $\#models \leftarrow$ Run $allSAT$ solver on $P_=(h)$
  $M = M \cup \{\#models\}$
  $S_{input} \leftarrow S_{input} \setminus \{s\}$
end

Partition for program P is $M = \{$1 model$\}\{$3 models$\}$

Implementation: AQuA



Main features & constraints

- runs on subset of ANSI-C, without memory alloc, only integer secrets, no interactive input
- no annotations needed except cmdline options
- supports non-linear arithmetic and integer overflows
- Tool chain: CBMC, Spear, RelSat, C2D
- Computation easily distributed

Loops and Soundness

Bounded loop unrolling is a source of unsoundness: not all possible behaviours are considered.

```
l=0; while(l < h) { l++; }
```

$\Downarrow$

```
l=0; if(l < h) { l++; if(l < h) { l++; ...
```

All untreated inputs end up in a “sink state”.
Program above with 4 bit variables and 2 unrollings generates partition: $\{1\}\{1\}\{14\}$

Entropy can be over-approximated by distributing the sink state into singletons: $\{1\}\{1\}\underbrace{\{1\}\ldots\{1\}}_{14\text{x}}$

From C to SPEAR

```
int main() {
    int h1,h2,h3,l;
    l = h1+h2+h3;
}
```

CBMC translates C to SSA constraints

```
tmp11 == (h110 + h210)
l11 == (h310 + tmp11)
```

For loops are unrolled completely, *while loops* up to user defined iteration.
CBMC is *not* used for model checking here!

Generate $P_{\neq}$ by translating intermediate language above

$P_{\neq}$ in SPEAR Format

```
d l11__:i12 tmp11__:i12 l11:i12 tmp11:i12 ...
p = h310 0:i12 # secret initialisations
p = h210 0:i12
p = h110 0:i12
p ule h310 5:i12 # constraining domain
p ule h310__ 5:i12
..
c tmp11 + h110 h210 # self composed program
c l11 + h310 tmp11
c tmp11__ + h110__ h210__
c l11__ + h310__ tmp11__
p /= l11__ l11
```

$P_{\neq}$ in SPEAR Format

```
d l11__:i12 tmp11__:i12 l11:i12 tmp11:i12 ...
p = h310 0:i12 # secret initialisations
p = h210 0:i12
p = h110 0:i12
p ule h310 5:i12 # constraining domain
p ule h310__ 5:i12
..
c tmp11 + h110 h210 # self composed program
c l11 + h310 tmp11
c tmp11__ + h110__ h210__
c l11__ + h310__ tmp11__
p /= l11__ l11
# model found:
h110__=5, h210__=5, h310__=5, l11__=15
```

$P_{\neq}$ in SPEAR Format

```
d l11__:i12 tmp11__:i12 l11:i12 tmp11:i12 ...
p = h310 5:i12 # secret initialisations
p = h210 5:i12
p = h110 5:i12
p ule h310 5:i12 # constraining domain
p ule h310__ 5:i12
..
c tmp11 + h110 h210 # self composed program
c l11 + h310 tmp11
c tmp11__ + h110__ h210__
c l11__ + h310__ tmp11__
p /= l11__ l11
# blocking clauses to not find same solutions again
p /= l11__ 15:i12
```

$P_{=}$ in SPEAR Format

```
d l11__:i12 tmp11__:i12 l11:i12 tmp11:i12 ...
p = h310 ?:i12 # secret initialisations
p = h210 ?:i12
p = h110 ?:i12
p ule h310 5:i12 # constraining domain
p ule h310__ 5:i12
..
c tmp11 + h110 h210 # self composed program
c l11 + h310 tmp11
c tmp11__ + h110__ h210__
c l11__ + h310__ tmp11__
p = l11__ l11
```

translated to CNF and fed to model counters (relsat, c2d)

Estimating Entropy

Example: Sample S with 3 equivalence classes to get the partition on an input space of 7 bit (128 unique inputs).

$$\{5\}\{5\}\{6\} \qquad (\frac{5}{128}, \frac{5}{128}, \frac{6}{128})$$

Intuition: Estimate remaining number of equivalence classes proportional to the sample S and distribute remaining inputs equally.

3 eq. classes sampled with coverage $\frac{5+5+6}{128} = \frac{1}{8}$
Remaining $\frac{7}{8}$ of inputs (112) will be split in $7 * 3 = 21$ equivalence classes.

Computational Problems

The previous tools show that implementing a precise QIF analysis for secret sizes of more than a few bits is computationally unfeasible; roughly speaking this is because classical QIF computes the entropy of a random variable whose complexity is the same as computing all possible runs of the program.
So is QIF for real code possible?
Change the question: from “How much does it leaks?” to “Does it leak more than k?”.
We look for a lower bound to the channel capacity

Channel capacity

Channel capacity for P, i.e. the maximum possible leakage for P

```
if (password==guess) access=1; else
access=0;
```

Suppose the password is a 64 bits randomly chosen string.
Two blocks: $B_1 = \{\texttt{password}\}$ probability $\frac{1}{2^{64}}$,
$B_2 = \{\neq \texttt{password}\}$ $2^{64} - 1$ elements, probability $1 - \frac{1}{2^{64}}$.
Entropy = $3.46944695 \times 10^{-18}$: as expected a password check of a big password should leak very little.
But if $\frac{1}{2} = p(B_1) = p(B_2)$. Then the entropy = 1 which is the channel capacity, i.e.
Channel Capacity given two classes: $1 = \log_2(2)$.

Leakage for Linux Kernel Code

Heusser-Malacaria 2010: first application of these theories to real industrial code:

1. We can quantify leakage for real C Code, e.g. Linux Kernel Code: CVE (mitre.org) reported vulnerabilities
2. We can prove that the official patch eliminate the leaks

Demo

Experimental Results on C Code

Description	CVE Bulletin	LOC	$k^\star$	Patch Proof	$log_2(N)$
AppleTalk	CVE-2009-3002	237	64	✓	6 bit
tcf_fill_node	CVE-2009-3612	146	64	✓	6 bit
sigaltstack	CVE-2009-2847	199	128	✓	7 bit
cpuset†	CVE-2007-2875	63	64	×	6 bit
SRP getpass	–	93	8	✓	1 bit
login_unix	–	128	8	–	2 bit

Table 1: Experimental Results. $\star$ Number of unwind-

Quantifying Loss of Anonymity

Let's now consider protocols: Anonymity protocols:
Examples: a voting protocol (elect someone), an anonymous browsing protocol, anonymous messaging
Main difference with programs:
Non-determinacy, same input may produce different observations:
But the idea of leakage is the same: **difference** in the **uncertainty** about the secret h **before and after** observations O on the system:

$$H(h) - H(h|O) = I(h;O) \text{ (mutual information)}$$

Defining anonymity protocols

An anonymity protocol ϕ is a matrix where $\phi_{i,k}$ is the probability of observing o_k given the anonymous event h_i.

	o_1	o_2	$\ldots$	o_n
h_1	$\phi_{1,1}$	$\phi_{2,1}$	$\ldots$	$\phi_{n,1}$
h_2	$\phi_{1,1}$	$\phi_{2,1}$	$\ldots$	$\phi_{n,2}$
$\vdots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
h_m	$\phi_{1,m}$	$\phi_{2,m}$	$\ldots$	$\phi_{n,m}$

Table 2: Protocol matrix

Maximum loss of anonymity

Given an anonymity protocol how much information is leaked about confidential information?
e.g. in an election there is always some information leaked about voters preference:
e.g. if candidate A got $100\%A$ of the votes then we know exactly who Bob voted for...
We can study the problem of maximum loss of anonymity using a powerful mathematical technique: Lagrange Multipliers

Lagrange Multipliers

Suppose we want to maximize the following function:

$$10 - (x-5)^2 - (y-3)^2$$

Answer: minimize $(x-5)^2$ and $(y-3)^2$, i.e. $x = 5, y = 3$.
Suppose however we add the constraint $x + y = 1$. Then the above solution is no longer correct.
Try

$$10 - (x-5)^2 - (y-3)^2 + \lambda(x + y - 1)$$

the number λ is the Lagrange Multiplier

Lagrange Multipliers

Maximize

$$10 - (x-5)^2 - (y-3)^2 + \lambda(x+y-1)$$

Lagrange Technique:
Find the maximum of the function

$$10 - (x-5)^2 - (y-3)^2 + \lambda(x+y-1)$$

by differentiating on x, y *and* λ. So

$$-2x + 10 + \lambda = 0, \quad -2y + 6 + \lambda = 0, \quad x + y - 1 = 0$$

$$y + 2 + y = 1, \ i.e. \ y = -\frac{1}{2}, x = \frac{3}{2}, \ \lambda = -7$$

Maximum loss of anonymity

Applying the technique to our problem:
We want to maximize (over the secret h)

$$I(h; O) \text{ (mutual information)}$$

subject to some constraint; one always present constraint:
$\sum_i h_i = 1$

Channel Distribution

Theorem: The probabilities h_i maximizing $I(h;T)$ subject to the family of constraint $(C_k)_{k\in K}$ (where $C_k \equiv \sum_j h_j f_{j,k} = F_k$ and $f_{j,k}, F_k$ are constants) are given by solving in h_i the equations

$$\sum_{o_s \in \hat{O}_i} \phi_{i,s} \log(\frac{\phi_{i,s}}{o_s}) - d + \sum_k \lambda_k f_{i,k} \quad = \quad 0$$

(where $d = \frac{1}{\log 2}$)

Channel Capacity

Theorem: The channel capacity for $I(h;T)$ subject to the family of constraint $(C_k)_{k\in K}$ (where $C_k \equiv \sum_j h_j f_{j,k} = F_k$ and $f_{j,k}, F_k$ are constants) is given by

$$\sum_i h_i(d - \sum_k \lambda_k f_{i,k})$$

Moreover in the case of the single constraint $\sum_i h_i = 1$ the above simplify to

$$d - \lambda_0$$

Example: Binary symmetric channel

$h = o = \{0, 1\}$

$$\phi_{0,0} = \phi_{1,1} = 1 - p$$

$$\phi_{0,1} = \phi_{1,0} = p$$

Using $\sum_i h_i \phi_{k,i} = o_k$ we get

$$o_0 = (1-p)h_0 + ph_1 \quad o_1 = ph_0 + (1-p)h_1$$

Anonymity Protocols

(Chen-Malacaria) applied this technique to studying maximum loss of anonymity for anonymity protocols like Dyning Cryptographers, Crowds and Onion Routing.
The results extend previous work by Chastikokolakis-Palamidessi-Panangaden (it doesn't need assumption of symmetry about the protocol participants)

Conclusions

- Information Theory and the Lattice of Information are valuable tools in defining, understanding and measuring leakage of information.
- They allow for powerful reasoning principles e.g. loops.
- Automated tool built on SAT solving and model counting to calculate entropy: entropy estimators can improve performance
- Real code can be analysed (Basin-Kopf: cryptographic side-channels, Heusser-Malacaria: Linux kernel memory)