LIBQIF: A Quantitative Information Flow C++ Toolkit Library

Martinelli Fernán G.

Universidad Nacional de Río Cuarto
fmartinelli@dc.exa.unrc.edu.ar

Abstract. A fundamental concern in computer security is to control information flow, whether to protect confidential information from being leaked, or to protect trusted information from being tainted. A classic approach is to try to enforce non-interference. Unfortunately, achieving non-interference is often not possible, because often there is a correlation between secrets and observables, either by design or due to some physical feature of the computation (side channels). One promising approach to relaxing noninterference, is to develop a quantitative theory of information flow that allows us to reason about how much information is being leaked, thus paving the way to the possibility of tolerating small leaks. In this work, we aim at developing a quantitative information flow C++ toolkit library, implementing several algorithms from the areas of QIF (more specifically from four theories: Shannon Entropy, Min-Entropy, Guessing Entropy and G-Leakage) and Differential Privacy. The library can be used by academics to facilitate research in these areas, as well as by students as a learning tool. A primary use of the library is to compute QIF measures as well as to generate plots, useful for understanding their behavior. Moreover, the library allows users to compute optimal differentially private mechanisms, compare the utility of known mechanisms, compare the leakage of channels, compute gain functions that separate channels, and various other functionalities related to QIF.

Keywords:
QIF, Quantitative , Information Flow, C++ library.

1 Introduction

A fundamental concern in computer security is to control information flow, whether to protect confidential information from being leaked, or to protect trusted information from being tainted.

A classic approach is to try to enforce non-interference, which is a property stating that the observables (behavior, outputs) of a system are independent from the secrets. This means that an adversary cannot deduce anything about the secrets from the observables. Unfortunately, achieving non-interference is often not possible, because often there is a correlation between secrets and observables, either by design or due to some physical feature of the computation

(side channels). One promising approach for relaxing non-interference, is to develop a quantitative theory of information flow that allows us to reason about how much information is being leaked, thus paving the way to the possibility of tolerating small leaks.

The basis of such a theory is a measure of leakage. One of the most successful approaches is based on information theory, the idea being that a system is seen as a (noisy) channel, whose inputs corresponds to the secrets, and the outputs to the observables.

The contribution of this work consists on developing a quantitative information flow C++ toolkit library, implementing several algorithms from the areas of Quantitative Information Flow. In addition, we present a case study over the G-Leakage theory finding easily some interesting properties with this library.

2 Foundations of Quantitative Information Flow

The study of information theory started with Claude E. Shannon's work on the problem of coding messages to be transmitted through unreliable (or noisy) channels. A *communication channel* is a (physical) means through which information can be transmitted. The input is fed into the channel, but due to noise or any other problems that can occur during the transmission, the output of the channel may not reflect with fidelity the input. It is usual to describe the unreliable behavior of the channel in a probabilistic way. In the discrete (finite) case, if $X = \{x_1, x_2, \ldots, x_n\}$ represent the possible inputs for the channel, and $Y = \{y_1, y_2, \ldots, y_m\}$ represent the possible outputs, the channel's probabilistic behavior can be represented as a channel matrix $C_{n\times m}$ where each element $C_{i,j}$ ($1 \leq i \leq n$, $1 \leq j \leq m$) is defined as the probability of the channel outputting b_j when the input is a_i. In this way, we can see the input and output as two correlated random variables linked by the channel's probabilistic behavior [1].

A unique feature of information theory is its use of a numerical measure of the amount of information gained when the contents of a message are learned. More specifically, information theory reasons about the degree of uncertainty of a certain random variable, and the amount of information that it can reveal about another random variable. Among the tools provided by information theory there are concepts as *entropy*, *conditional entropy*, *mutual information* and *channel capacity*.

Several works in the literature use an information theoretic approach to model the problem of information flow and define the leakage in a quantitative way, as for example [ZB05,CHM05,Mal07,MC08,MNS03,MNCM03,CPP08a]. The idea is to model the computational system as an *information theoretic channel*. The input represents the secret, the output represents the observable, and the correlation between the input and output (*mutual information*) represents the information leakage. The worst case leakage corresponds then to the *capacity* of

[1] Note that we are assuming that channels are *loseless*, since the rows are probability distributions instead of sub-probability distributions.

the channel, which is by definition the maximum mutual information that can be obtained by varying the input distribution.

In the works mentioned above, the notion of mutual information is based on *Shannon entropy*, which (because of its mathematical properties) is the most established measure of uncertainty. From the security point of view, this measure corresponds to a particular model of attack and a particular way of estimating the security threat (vulnerability of the secret). Other notions have been considered, and argued to be more appropriate for security in certain scenarios. These include: *min-entropy* [R61,Smi09], *Bayes risk* [CT06,CPP08b], *guessing entropy* [Mas94], and *marginal guesswork* [Pli00].

Whatever definition of uncertainty (i.e. vulnerability) we want to adopt, the notion of leakage is inherent to the system and can be expressed in a uniform way as the difference between the initial uncertainty, i.e. the degree of ignorance about the secret *before* we run the system, and the remaining uncertainty, i.e. the degree of ignorance about the secret *after* we run the system and observe its outcome. Following the principle advocated by Smith [Smi09], and by many others:

$$\begin{aligned} \textit{information leakage} \;=\; & \textit{initial uncertainty} \\ & - \\ & \textit{remaining uncertainty} \end{aligned} \tag{1}$$

In (1), the initial uncertainty depends solely on the input distribution, aka the *a priori distribution* or *prior*. Intuitively, the more uniform it is, the less we know about the secret (in the probabilistic sense). After we run the system, if there is a probabilistic correlation between input and output, then the observation of the output should increase our knowledge of the secret. This is determined by the fact that the distribution on the input changes. In fact we can update the probability of each input with the corresponding conditional probability of the same input, given the output. The new distribution is called the *a posteriori distribution*. In case the input and output are independent, then the a priori and the a posteriori distributions coincide and the knowledge should remain the same. We will use the attributes a priori (or prior) and a posteriori to refer to before and after the observation of the output, respectively.

The above intuitions should be reflected by any reasonable notion of uncertainty: it should be higher on more uniform distributions, and it should decrease or remain equal with the observation of related events.

Probability Distribution Vector

The notion of Probability Distribution Vector(PDV) is a vector which satisfies that:

- all the elements are greater than or equal to zero.
- the sum of all the elements is equal to 1.

We will use the notation $\pi(x)$ for talk about the Probability Distribution over the random variable $X = \{x_1, .., x_n\}$.

Channel
A channel is a Probability Distribution Matrix which satisfies that:

- all the elements are greater than or equal to zero.
- the sum of all the elements of each row is equal to 1.

We will use the notation $C[x, y]$for talk about the Joint Probability Distribution over the random variable formed like $X.Y(i.eC[x, y] = P(y|x)$ where $X = \{x_1, .., x_n\}$ and $Y = \{y_1, .., y_m\}$).

Entropy
The notion of entropy should be chosen according to the model of attacker, and to the way we estimate the success of the attack. Normally, the entropy of a random variable represents its uncertainty, hence the vulnerability is anti-monotonic on the entropy.

There exists two notions of entropy: prior entropy and conditional entropy.

Vulnerability
Vulnerability is a measure of how easy is for an attacker to discover the secret. The concept of vulnerability is the opposite of uncertainty.

There exists two notions of vulnerability:

- prior vulnerability
  We will use the notation $V(\pi)$ to refer to the vulnerability.
- conditional vulnerability
  We will use the notation $V(\pi, C)$ to refer to the conditional vulnerability.

Leakage
Leakage is a measure of how much information can the attacker know about the secrets with a specific probability distribution of the secrets and channel matrix.

The notion of leakage can be expressed as the difference between the initial uncertainty about the secret and the remaining uncertainty:

$$\text{information leakage} = L(\pi, C) = \text{initial uncertainty - remaining uncertainty.}$$

Capacity
The notion of channel capacity or maximum leakage $\text{ML}(\pi, C)$ is the maximum amount of information that can be known by the attacker in the worse case with a specific channel matrix. i.e. over all the possible probability distributions.

2.1 Shannon Entropy Leakage

The (Shannon) *entropy* of X is defined as $H(X) = -\sum_{\mathcal{X}} \pi(x) \log \pi(x)$.

The entropy measures the uncertainty of X. It takes its minimum value $H(X) = 0$ when $\pi(x)$ is a point mass (also called delta of Dirac). The maximum value $H(X) = \log |\mathcal{X}|$ is obtained when $\pi(x)$ is the uniform distribution. Usually the base of the logarithm is set to be 2 and the entropy is measured in *bits*. Roughly speaking, m bits of entropy means that we have 2^m values to choose from, assuming a uniform distribution.

The *conditional entropy* of X given Y is defined as

$$H(X \mid Y) = \sum_{\mathcal{Y}} \pi(y)\ H(X \mid Y = y)$$

where

$$H(X \mid Y = y) = -\sum_{\mathcal{X}} C[x, y] \log\ C[x, y]$$

The famous *Channel Coding Theorem* by Shannon relates the capacity of the channel to its maximum transmission rate. In brief, the channel capacity is a tight upper bound for the maximum rate by which information can be reliably transmitted using the channel. Given an acceptable probability of error ξ, there is a natural number n and a coding for which n uses of the channel will result in messages being transmitted with at most the acceptable probability of error ξ.

The algorithm used to calculate the capacity of the channel on the Shannon Approach is called Blahut-Arimoto algorithm. It was presented in [Ip99,Bla72,Ari72]. Lightly, the authors have reformulated the problem as a *convex optimization problem* after proving that Shannon Entropy is a concave function.

Meaning in security: To explain what $H(X)$ represents from the security point of view, consider a partition $\{\mathcal{X}_i\}_{i \in I}$ of $\mathcal{X}$. The adversary is allowed to ask questions of the form does $X \in \mathcal{X}_i$? according to some strategy. Let $n(x)$ be the number of questions that are needed to determine the value of x, when $X = x$. Then $H(X)$ represents the lower bound to the expected value of $n(\cdot)$, with respect to all possible partitions and strategies of the adversary [Pli00,KB07].

2.2 Min-Entropy Leakage

In [Rényi61], Rényi introduced a one-parameter family of entropy measures, intended as a generalization of Shannon entropy.

Various researchers, including Cachin [Cac97], have considered the following definition:

$$H_{\alpha}^{Cachin}(X \mid Y) = \sum_{y\, \in\, Y} \pi(y)\ H_{\alpha}(X \mid Y = y)$$

which, as $\alpha \to \infty$, becomes

$$H_{\infty}^{Cachin}(X \mid Y) = -\sum_{y\, \in\, Y} \pi(y)\ \log \max_{x \in \mathcal{X}} C[x, y] \tag{2}$$

An alternative proposal for $H_{\infty}(\cdot \mid \cdot)$ came from Smith [Smi09][2]:

$$H_{\infty}^{Smith}(X \mid Y)\ =\ -\log \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}}\ C[x, y] \tag{3}$$

[2] The same formulation had been already used by Dodis et al. in [DORS04], and Smith proposed it independently. Since it is Smith's work on the subject that motivates the approach used in this thesis, we opt to refer to this formulation as Smith's.

Meaning in security: The min-entropy can be related to a model of adversary who is allowed to ask exactly one question, which must be of the form is $X = x$? (one-try attacks). More precisely, the min-entropy $H_\infty(X)$ represents the probability of success for this kind of attack and with the best strategy, which consists, of course, in choosing the x with the maximum probability.

As for $H_\infty(X \mid Y)$ and $I_\infty(X;Y)$, the most interesting versions in terms of security seem to be those of Smith. In fact, in this thesis we adopt his approach to information leakage, and we will, from now on, use the following notation:

- $H_\infty(X \mid Y)$ stands for $H_\infty^{Smith}(X \mid Y)$ and is referred to as *conditional min-entropy*;
- $I_\infty(X;Y)$ stands for $I_\infty^{Smith}(X;Y)$ and is referred to as *min-entropy leakage*.

In fact, the conditional min-entropy $H_\infty(X \mid Y)$ represents the log of the inverse of the (expected value of the) probability that the same kind of adversary succeeds in guessing the value of X *a posteriori*, i.e. after observing the result of Y. The complement of this probability is also known as *probability of error* or *Bayes risk*. Since in general Y and X are correlated, observing Y increases the probability of success. In fact, we can prove formally that $H_\infty(X \mid Y) \leq H_\infty(X)$, with equality if X and Y are independent. The min-entropy leakage $I_\infty(X;Y)$ corresponds to the *ratio* between the probabilities of success a priori and a posteriori, which is a natural notion of leakage. Here $I_\infty(X;Y)$ is in the format of 1, but the difference becomes a ratio due to the presence of the logarithms. Note that $I_\infty(X;Y) \geq 0$, which seems desirable for a good notion of leakage. It has been proven in [BCP09] that ML_∞ is obtained at the uniform distribution, and that it is equal to the sum of the maxima of each column in the channel matrix, i.e. $\mathrm{ML}_\infty = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} C[x,y]$.

2.3 G-Leakage

In a more recent approach, g-leakage [ACPS12], the benefit that an adversary derives from a certain guess about a secret is specified using a gain function g. Gain functions allow a wide variety of operational scenarios to be modelled, including those where the adversary benefits from guessing a value close to the secret, guessing a part of the secret, guessing a property of the secret, or guessing the secret within some number of tries.

They now adapt the definition of vulnerability to take account of the gain function:

Given gain function g and prior π, the *prior g-vulnerability* is

$$V_g(\pi) = \max_{w \in W} \sum_{x \in X} \pi[x] g(w,x).$$

The idea is that adversary should make a guess w that maximizes the expected gain; They therefore take the weighted average of $g(w,x)$, for every possible value x of X.[3]

[3] They remark that our assumption that gain values are between 0 and 1 is unimportant. Allowing g to return a value in $[0,a]$, for some constant a, just scales all g-vulnerabilities by a factor of a and therefore has no effect on g-leakage.

Given gain function g, prior π, and channel C, the *posterior g-vulnerability* is

$V_g(\pi, C)$
$= \sum_{y \in Y} \max_{w \in W} \sum_{x \in X} \pi[x] C[x, y] g(w, x)$
$= \sum_{y \in Y} \max_{w \in W} \sum_{x \in X} p(x, y) g(w, x)$
$= \sum_{y \in Y} p(y) V_g(p_{X|y})$

The authors have defined g-entropy, g-leakage, and g-capacity : $H_g(\pi) = -\log V_g(\pi)$
$H_g(\pi, C) = -\log V_g(\pi, C)$
$L_g(\pi, C) = H_g(\pi) - H_g(\pi, C) = \log \frac{V_g(\pi, C)}{V_g(\pi)}$
$ML_g(C) = \sup_\pi L_g(\pi, C)$

Gain Functions A Gain Function is a Matrix which will be used for modelling the benefits of the attacker. So, the rows of the matrix will be elements of a set $W = \{w_1, .., w_m\}$ which are the possible guesses. And the columns of this matrix are elements of $X = \{x_1, .., x_n\}$. i.e. elements of the inputs set.

We will use the notation $g[w, x]$ to talk about the benefit of guess w when the secret is x.

2.4 Guessing Entropy Leakage

The notion of guessing entropy was introduced by Massey in [Mas94]. Let us assume, for simplicity, that the elements of $\mathcal{X}$ are ordered by decreasing probabilities, i.e. if $1 \leq i < j \leq n$ then $p(x_i) \geq p(x_j)$. Then the guessing entropy is defined as follows: $H_G(X) = \sum_{1 \leq i \leq |\mathcal{X}|} i\, \pi(x_i)$

Massey did not define the notion of conditional guessing entropy. In some works, like [Cac97,KB07], it is defined analogously to 2.1: $H_G(X \mid Y) = \sum_{y \in \mathcal{Y}} \pi(y)\ H_G(X \mid Y = y)$

Meaning in security: Guessing entropy represents an adversary who is allowed to ask repeatedly questions of the form is $X = x?$. More precisely, $H_G(X)$ represents the expected number of questions that the adversary needs to ask to determine the value of X, assuming that he follows the best strategy, which consists, of course, in choosing the x's in order of decreasing probability.

$H_G(X \mid Y)$ represents the expected number of questions *a posteriori*, i.e. after observing the value of Y and reordering the queries according to the updated probabilities (i.e. the queries will be chosen in order of decreasing a posteriori probabilities).

2.5 Differential Privacy

In the area of statistical databases, one of the most prominent approaches for protecting an individuals privacy when releasing aggregate information is that of Differential Privacy [Dwo06]. This notion ensures that changes to a single

individuals value have negligible effect on the querys outcome. This notion is closely related to information flow [AACP11,BK11] since differentially private mechanisms can be seen as information theoretic channels, and bounds can be obtained for the information leakage of those channels.

The notion of *differential privacy*, due to Dwork [Dwo06,DL09,Dwo10,Dwo11], is a proposal to control the risk of violating privacy for both kinds of threats described above (value and participation). The idea is to say that a randomized function $\mathcal{K}$ satisfies ϵ-differential privacy (for some $\epsilon > 0$) if the ratio between the probabilities that two adjacent databases give a certain answer is bound by e^{ϵ}, where by adjacent we mean that the databases differ in only one individual (either for the value of an individual or for the presence/absence of an individual). The notion of differential privacy was developed to be independent of the *side (or auxiliary) information* the user can have about the database, and how it can affect his knowledge about the database before posing the query. This information can come from external sources (e.g. newspapers, common knowledge, etc), but does not affect the guarantees assured by differential privacy.

3 Developing a library for QIF study

3.1 Goal

The LIBQIF library was conceived as a C++ toolkit, due to the portability and high support of the C++ community. Also, we needed an efficient language for calculating the QIF measures.

LIBQIF aims to provide a simple interface for hiding the implementation features and allowing the users to create QIF examples, to calculate automatically measures, and to plot functions, among others. Without the necessity of knowing how to use plotters, how to implement matrix and vectors in C++, how to represent graphs, or how to implement the known algorithms.

A primary use of the library is to compute QIF measures as well as to generate plots, useful for understanding their behavior. This library aims to cover the QIF approaches like Shannon Entropy Leakage, Min-Entropy Leakage, G-Leakage and Guessing Entropy Leakage. Moreover, the library will allow to compute optimal differentially private mechanisms, compare the utility of known mechanisms, compare the leakage of channels, compute gain functions that separate channels, and various other functionalities related to QIF.

This work had the goal of extending the G-Leakage theory with an efficient algorithm for calculate the G-Capacity, i.e. the capacity of the channel under the G-Leakage theory. In Section 4 we discuss the difficulties that we have encountered in the attempt to achieve this goal.

3.2 Implementation features

This section gives an explanation about the library implementation features.

In QIF, the learning examples are 3x3 matrices approximately, but, when the researcher talks about databases, or maybe more realistic QIF cases, the number

is bigger. So, one important feature is that LIBQIF should be fast. And this is why C++ was chosen as programming language. And also, because C++ is very contributed and supported by free software libraries.

LIBQIF implements Channel matrix, Gain function and probabilistic distribution vector using the implementation of **matrices and vectors** from the **Armadillo library** (more specifically the classes mat and vec). Armadillo is an optimized library for using matrices and vectors.

LIBQIF, is well documented using **Doxygen**. Doxygen is the tool used to automatically document the project from the sources. The **LIBQIF documentation** can be founded in html/index.html.

LIBQIF implements graphs by a simple class because advanced graphs are not necessary. So the **class Graph** is a simple adjacency matrix.

For some algorithms is necessary be able to solve **linear programming** problems. This is why there exist a class called LinearProgram, which defines two functions that are the interface of how to write this problems and translate them for solving them using a library called **GLPK**.

For the testing of LIBQIF, a library named **GTEST** was utilized for unit **testing**.

Looking for **plotter engines**, there are some ones like **Scilab** (free software), Matlab, Maple or GNUPlot. The class that implements the interaction with this plotter engines is EntropyModel. And it was designed for supporting all of them but just Scilab engine is implemented at the first version of LIBQIF. The reason because of choosing Scilab was that it is relatively easy with the scilab-call API and it is free software like LIBQIF.

One interesting feature that is not immediately observable is the fact that LIBQIF is working with the **generic arithmetic precision of the computer**, and sometimes this precision can change the results on numeric/statistical calculus. One interesting extension of LIBQIF would be that LIBQIF could work with precise arithmetic.

On Appendix A there is a summary of the QIF theories implemented on the LIBQIF library.

4 G-Leakage: A case study

The second part of this work consists on extending the G-Leakage theory with a method for computing the capacity of the channel (g-capacity). [4]

The G-Leakage theory was explained like a generalization of the particular min-entropy case. Unfortunately, the generalization has not the same properties.

A property of min-capacity that makes it easy to compute is that it is always realized on a uniform prior. Alvim, Chatzikokolakis, Palamidessi and Smith have found in [ACPS12], however, that this does *not* hold for g-capacity.

[4] Remind that the capacity of the channel is the maximum leakage over all the priors.

Lets remind their example:

	y_1	y_2
x_1	0.6	0.4
x_2	0	1
x_3	0	1

(4)

Now suppose we use the following metric-induced gain function g_d:

g_d	x_1	x_2	x_3
x_1	1	0	0
x_2	0	1	0.98
x_3	0	0.98	1

Consider 0 leakage channel C_4 above and its gain function g_d. Under a uniform prior π, we compute that $V_{g_d}(\pi) = 0.66$, $p_Y = (0.2, 0.8)$, $V_{g_d}(p_{X|y_1}) = 1$, $V_{g_d}(p_{X|y_2}) = 0.825$, and $V_{g_d}(\pi, C_4) = 0.86$, giving $L_{g_d}(\pi, C_4) = 0.3819$.

Now if we consider the prior $\pi' = (0.5, 0.5, 0)$, we find that $V_{g_d}(\pi') = 0.5$, $p_Y = (0.3, 0.7)$, $V_{g_d}(p_{X|y_1}) = 1$, $V_{g_d}(p_{X|y_2}) = \frac{5}{7}$, and $V_{g_d}(\pi', C_4) = 0.8$, which gives $L_{g_d}(\pi', C_4) = \log 1.6 \approx 0.6781$. Hence the g_d-capacity of 0 leakage channel is *not* realized on a uniform distribution.

Notice here that $\log 1.6$ is also 0 leakage channel's min-capacity. Hence, by the "Miracle" Theorem [5], It is known that $\log 1.6$ must in fact be its g_d-capacity, realized on π'. But, so far, the authors have not found a general technique for calculating g-capacity; this is the motivation of the second part of this paper.

We have utilized the LIBQIF library to try to develop a formal method to compute the channel capacity on the G-Leakage theory.

As we have seen in the Section 2.1, on Shannon Entropy Leakage theory, the channel capacity is calculated by an algorithm called Blahut-Arimoto Algorithm. It consists on converting the problem to a convex optimization problem. But, before we can apply the same idea, we will need prove that G-Leakage is a concave function.

In the LIBQIF code Appendix D, we have started looking for properties that allow us to see if G-Leakage satisfies what we need.

In the code, we have remarked a particular case that gave us the first observation.

4.1 A first observation on G-Leakage

The G-Leakage theory is a generalization of Min-Entropy Leakage. The authors showed examples where the uniform prior (that in Min-Entropy Leakage gives the maximum Leakage) is not the probability distribution that maximizes the G-Leakage.

One first thing that should be interesting to know is if the probability distribution that minimizes $V(\pi)$ is the same one that maximizes $\log \frac{V(C,\pi)}{V(\pi)}$.

[5] For any channel C and gain function g, $ML_g(C) \leq ML(C)$.

And this is not true. The following is an example of this.

G:

G	x1	x2	x3
x1	1	0.5	0
x2	0.5	1	0.5
x3	0	0.5	1

C:

	y1	y2
x1	0.3	0.7
x2	0.7	0.3
x3	0.3	0.7

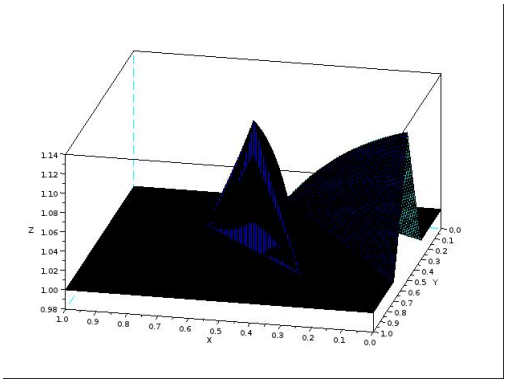

Fig. 1. Example on G-Leakage.
[a]

[a] In the graphic, the log is not applied. This way, the results are easier to read. With the log nothing change because the log is a monotonic function.

Here, vectors (0,0.5,0.5) or (0.5,0.5,0) give the maximum leakage 0.125. The prior vulnerability with (0,0.5,0.5) is 0.75.

The minimum prior vulnerability of 0.5 is achieved with the vector (0.5,0,0.5).

Remark: So, the probability distribution that minimizes $V(\pi)$ is *not* the same one that maximizes $\log \frac{V(C,\pi)}{V(\pi)}$. This means that on G-Leakage there are not good properties like on Min-Entropy to simplify the formula. [6]

4.2 On a method to calculate the G-Capacity

To find an iterative algorithm to calculate G-Capacity (like the one used for Shannon Entropy that uses **convex optimization problems**) we must prove that G-Leakage is a concave function or maybe a quasi-concave function.

concave function:

$$L_g(\lambda * \pi + (1-\lambda) * \pi', C) \geq \lambda * L_g(\pi, C) + (1-\lambda) * L_g(\pi', C)$$

quasi-concave function:

[6] We have used the LIBQIF code at Appendix D for computing this measures and generating the plots.

$$L_g(\lambda * \pi + (1-\lambda) * \pi', C) \geq \min(L_g(\pi, C), L_g(\pi', C))$$

Again, G-Leakage does not satisfy this definitions. The example 1 is not a quasi-concave function. So, it is not a concave function either.

Here,vectors (0,0.5,0.5) or (0.5,0.5,0) give the maximum leakage 0.125. With $\lambda = 0.5$ the new vector is (0.25,0.5,0.25) and the leakage with this probability distribution is 0. So, this **G-Function example does not satisfy the quasi-concave property**.

Moreover, we can prove that the special case of G_{id} **Min-Entropy is not quasi-concave**.

G:

G_{id}	x1	x2	x3
x1	1	0	0
x2	0	1	0
x3	0	0	1

C:

	y1	y2
x1	0.3	0.7
x2	0.7	0.3
x3	0.3	0.7

In Example 2, vectors (1/3,1/3,1/3), (0,0.5,0.5) or (0.5,0.5,0) gives the maximum leakage 0.336. With $\lambda = 0.5$ one possible new vector is (0.25,0.5,0.25), and the leakage with this probability distribution is 0.048.

Remark: So, this G-Function example does not satisfy the quasy-concavity property. [7]

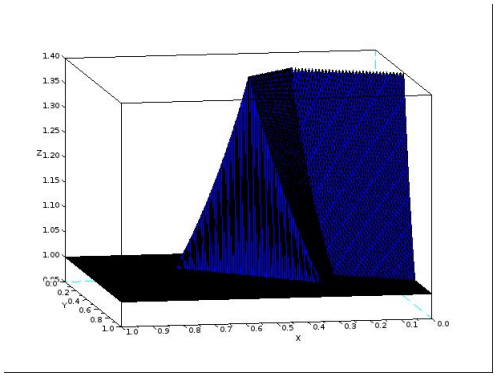

Fig. 2. Example looking for the G-Capacity.
[a]

[a] In the graphic, the log is not applied.This way, the results are easier to read. With the log nothing change because the log is a monotonic function.

Remark: It is possible to show that, by adding columns to the channel matrix C, the G_{id} can have a local maxima. So, it is not possible to use this kind of algorithms either.

[7] We have used the LIBQIF code at Appendix E for computing this measures and generating the plots.

Finally: We could not find the way to optimize the leakage under the G-Leakage theory. Then, G-Capacity remains an area for future study.

5 Conclusions and future work

The end result of developing LIBQIF is a C++ toolkit library that simplifies to researchers the process of plotting and calculating QIF measures without using mathematical programs like Matlab, Scilab, etc. And just with a basic user level of the C++ language. The LIBQIF syntax is really near to the QIF research concepts.

The LIBQIF library can be downloaded from the repository:

$$https://github.com/fmartinelli/libqif/tree/master/bin$$

For developers that want to contribute with the project can get up the sources from:

$$https://github.com/fmartinelli/libqif$$

The documentation of the library can be found with the libqif.tar.gz inside the directory docs. More specifically in the file index.html.

LIBQIF library supports the theory of Shannon Entropy, Min-Entropy Leakage, Guessing Entropy, G-Leakage and Differential Privacy.

This library is free software; So, the users can redistribute it and/or modify it under the terms of the GNU Lesser General Public License.

We have showed examples of how to use the LIBQIF library, and more interestingly, we have utilized the library for researching on the G-Leakage theory.

Unfortunately, the attempts to find a method to calculate the g-capacity failed. But, we have found interesting optimization features over min-entropy and g-leakage.

Some ones of these discoveries are that min-entropy is not quasi-concave (consequently it is not concave). So, it can not be written like a convex optimization problem to use an iterative algorithm similarly to Shannon entropy.

G-Leakage is a generalization of min-entropy, so, g-leakage can not be written like a convex optimization problem either.

As future way to follow in researching on a method to calculate the g-capacity maybe it is possible try to use non-convex problem solving algorithms.

LIBQIF has helped us to find easily counterexamples. But, it has some limitations for the moment. There is just one plotter engine (SciLab) implemented at the first version.

Future work on LIBQIF will be extend the implementation of plotter engines allowing to the users to choose between SciLab, Maple, GNU-Plot and MATLAB.

Another interesting extension of LIBQIF would be that LIBQIF could work with precise arithmetic. It is not immediately observable the fact that LIBQIF is working with the generic arithmetic precision of the computer, and sometimes this precision can change the results on numeric/statistical calculus.

LIBQIF will support precise arithmetic as well in the future.

References

[AACP11] Mário Alvim, S., Miguel Andres, E., Konstantinos Chatzikokolakis, and Catuscia Palamidessi. On the relation between Differential Privacy and Quantitative Information Flow. In Jiri Sgall Luca Aceto, Monika Henzinger, editor, *38th International Colloquium on Automata, Languages and Programming - ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 60–76, Zurich, Switzerland, 2011. Springer.

[ACPS12] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 265–279, 2012.

[Ari72] Suguru Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.

[BCP09] Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Quantitative notions of leakage for one-try attacks. In *Proceedings of the 25th Conf. on Mathematical Foundations of Programming Semantics*, volume 249 of *Electronic Notes in Theoretical Computer Science*, pages 75–91. Elsevier B.V., 2009.

[BK11] Gilles Barthe and Boris Köpf. Information-theoretic bounds for differentially private mechanisms. In *CSF*, pages 191–204, 2011.

[Bla72] Richard E. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4):460–473, 1972.

[Cac97] Christian Cachin. *Entropy Measures and Unconditional Security in Cryptography*. PhD thesis, ETH Zürich, 1997. Reprint as vol. 1 of *ETH Series in Information Security and Cryptography*, ISBN 3-89649-185-7, Hartung-Gorre Verlag, Konstanz, 1997.

[CHM05] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative information flow, relations and polymorphic types. *J. of Logic and Computation*, 18(2):181–199, 2005.

[CPP08a] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Inf. and Comp.*, 206(2–4):378–401, 2008.

[CPP08b] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. On the Bayes risk in information-hiding protocols. *Journal of Computer Security*, 16(5):531–571, 2008.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., second edition, 2006.

[DL09] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proc. of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 371–380. ACM, 2009.

[DORS04] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. technical report 2003/235, cryptology eprint archive, http://eprint.iacr.org, 2006. previous version appeared at eurocrypt 2004. In *34 [DRS07] [DS05] [EHMS00] [FJ01] Yevgeniy Dodis, Leonid Reyzin, and Adam*, pages 79–100. Springer-Verlag, 2004.

[Dwo06] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming, 33rd Int. Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proc., Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

[Dwo10] Cynthia Dwork. Differential privacy in new settings. In *Proc. of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 174–183. SIAM, 2010.

[Dwo11] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–96, 2011.

[Ip99] Lawrence Ip. The blahut-arimoto algorithm for the calculation of the capacity of a discrete memoryless channel. Technical report, Technical Report, Berkeley Uni, 1999.

[KB07] Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security (CCS 2007)*, pages 286–296. ACM, 2007.

[Mal07] Pasquale Malacaria. Assessing security threats of looping constructs. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 225–235. ACM, 2007.

[Mas94] Massey. Guessing and entropy. In *Proceedings of the IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[MC08] Pasquale Malacaria and Han Chen. Lagrange multipliers and maximum information leakage in different observational models. In Úlfar Erlingsson and Marco Pistoia, editor, *Proceedings of the 2008 Workshop on Programming Languages and Analysis for Security (PLAS 2008)*, pages 135–146, Tucson, AZ, USA, June 2008. ACM.

[MNCM03] Ira S. Moskowitz, Richard E. Newman, Daniel P. Crepeau, and Allen R. Miller. Covert channels and anonymizing networks. In *Workshop on Privacy in the Electronic Society 2003*, pages 79–88, 2003.

[MNS03] Ira S. Moskowitz, Richard E. Newman, and Paul F. Syverson. Quasi-anonymous channels. In *Proc. of CNIS*, pages 126–131. IASTED, 2003.

[Pli00] Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. In *Proceedings of INDOCRYPT: International Conference in Cryptology in India*, number 1977 in Lecture Notes in Computer Science, pages 67–79. Springer-Verlag, 2000.

[Ré61] Alfréd Rényi. On Measures of Entropy and Information. In *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics, and Probability*, pages 547–561, 1961.

[Smi09] Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *Proceedings of the 12th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2009)*, volume 5504 of *LNCS*, pages 288–302, York, UK, 2009. Springer.

[ZB05] Ye Zhu and Riccardo Bettati. Anonymity vs. information leakage in anonymity systems. In *Proc. of ICDCS*, pages 514–524. IEEE Computer Society, 2005.

A Summary of QIF Theories

Shannon Entropy Leakage

1. Entropy: $H(\pi) = -\sum_{\mathcal{X}} \pi(x) \log \pi(x)$
2. Conditional Entropy: $H(\pi, C) = -\sum_{\mathcal{X}} \pi(x) \, (\sum_{\mathcal{Y}} C[x,y] \log \, C[x,y])$
3. Capacity: $\mathrm{ML}(C) = \max_{\pi} L(\pi, C)$ to compute this measure we use the Blahut-Arimoto Algorithm:
   The BA algorithm consists of alternately finding the optimal ϕ for a given π and the the optimal π for a given ϕ. Steps:
   (a) $\pi^1 =$ initial probability vector. And then iterate steps 3b and 3c t times.
   (b) Maximize $J(\pi^t, C, \phi)$ with respect to ϕ. So, $\phi_j^t = \frac{C[x,y] \, \pi^t(x)}{\sum_{k \in \mathcal{X}} C[k,y] \, \pi^t(k)}$.
   (c) Maximize $J(\pi, C, \phi^t)$ with respect to π. And this implies that, $\pi^{t+1}(x) = \frac{r^t(x)}{\sum_{k \in \mathcal{X}} r^t(k)}$. Where $r^t(x) = \exp \sum_{\mathcal{Y}} C[x,y] \, log \, \phi^t(x,y)$

Min-Entropy Leakage

4. Vulnerability: $V(\pi) = \max_{\mathcal{X}} \pi(x)$
5. Conditional Vulnerability: $V(\pi, C) = \sum_{\mathcal{Y}} \max_{\mathcal{X}} \pi[x] \, C[x,y]$
6. Entropy: $H(\pi) = -\log V(\pi)$
7. Conditional Entropy: $H(\pi, C) = -\log V(\pi, C)$
8. Capacity: $\mathrm{ML}(C) = \log \sum_{Y} \max_{X} C[x,y]$
   and it is realized on a uniform prior π.

G-Leakage

9. Vulnerability: $V(\pi) = \max_{\mathcal{W}} \sum_{\mathcal{X}} \pi[x] \, g(w,x)$
10. Conditional Vulnerability: $V(\pi, C) = \sum_{\mathcal{Y}} \max_{\mathcal{W}} \sum_{\mathcal{X}} \pi(x) \, C[x,y] \, g(w,x)$
11. Entropy: $H(\pi) = -\log V(\pi)$
12. Conditional Entropy: $H(\pi, C) = -\log V(\pi, C)$

Guessing Entropy Leakage

13. Entropy: $H(\pi) = \mathrm{G}(\mathrm{sort}(\pi))$
    where: $G(v) = \sum_{x \in [1..|v|]} x * v(x)$

14. Conditional Entropy: $H(\pi, C) = \sum_{Y} G(sort(v_y))$
    where v_y is a vector depending of y: $v_y = [\pi(1) * C[1, y], ..., \pi(|X|) * C[|X|, y]]$

Differential Privacy

15. Algorithm: is_differential_private(Channel C,Graph G)

```
for each (x, x') in edges(G):
    for each y in Y:
        if C[x,y] > e^epsilon * C[x',y]:
            return false
return true
```

B The LIBQIF License

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C Gain Function Examples

C.1 Identity Gain Function

A special gain function case is the identity which satisfies that

$$g[w, x] = \begin{cases} 1 & \text{if } w == x \\ 0 & \text{otherwise} \end{cases}$$

Per example the following is an identity gain function with size 3.

g:

g_{id}	x1	x2	x3
w1	1	0	0
w2	0	1	0
w3	0	0	1

The g_{id} (identity gain function) is the special case of G-Leakage = Min-Entropy Leakage.

C.2 k-tries Gain Function

Per example the following is an k-tries gain function with size 3.

g:

$g_{2-tries}$	x1	x2	x3
$w1 = \{x1, x2\}$	1	1	0
$w2 = \{x1, x3\}$	1	0	1
$w3 = \{x2, x3\}$	0	1	1

C.3 Gain Function from metrics

Exploring other gain functions, one quite natural kind of structure that X may exhibit is a notion of distance between secrets. That is, there may be a metric d on X , which is a function

$$d : X \times X \to [0, \infty)$$

Given a metric d, we can first form a normalized metric $\bar{d}$ by dividing all distances by the maximum value of d, and then we can define a gain function g_d by

$$g_d(w, x) = 1 - \bar{d}(w, x).$$

Per example the following is an distance gain function with size 3.

g:

g_d	x1	x2	x3
w1	1	0.5	0
w2	0.5	1	0.5
w3	0	0.5	1

where

$$d(w,x) = \begin{cases} 0 & \text{if } w == x \\ 0.5 & \text{if } w == x \pm 1 \\ 1 & \text{otherwise} \end{cases}$$

D LIBQIF Case Study Code

```
#include <iostream>
#include <string>
#include "GLeakage.h"

int main()
{
    std::cout << "Using LIBQIF Library looking for properties" << std::endl;

    std::string rand = "0.3 0.7; 0.7 0.3; 0.3 0.7";
    std::string balanced = "0.25 0.5 0.25; 0 1 0; 0.5 0 0.5";
    std::string metrics = "1 0.5 0; 0.5 1 0.5; 0 0.5 1"
    std::string id = "1 0 0; 0 1 0; 0 0 1";
    std::string k_tries = "1 1 0; 1 0 1; 0 1 1";

    Channel C_rand= Channel(rand);
    Channel C_balanced= Channel(balanced);
    Channel C_id= Channel(id);

    Gain g_id=Gain(id);
    Gain g_metrics=Gain(metrics);
    Gain g_k_tries=Gain(k_tries);

    //GLeakage
    GLeakage gl1= GLeakage(C_rand,g_id);
    GLeakage gl2= GLeakage(C_balanced,g_id);
    GLeakage gl3= GLeakage(C_id,g_id);

    GLeakage gl4= GLeakage(C_rand,g_metrics);
    GLeakage gl5= GLeakage(C_balanced,g_metrics);
    GLeakage gl6= GLeakage(C_id,g_metrics);

    GLeakage gl7= GLeakage(C_rand,g_k_tries);
    GLeakage gl8= GLeakage(C_balanced,g_k_tries);
    GLeakage gl9= GLeakage(C_id,g_k_tries);

    //ploting
    gl1.plot3d_leakage();
    gl2.plot3d_leakage();
```

```
    gl3.plot3d_leakage();

    gl4.plot3d_leakage(); //<-----

    gl5.plot3d_leakage();
    gl6.plot3d_leakage();
    gl7.plot3d_leakage();
    gl8.plot3d_leakage();
    gl9.plot3d_leakage();

}
```

plotting example code.

E LIBQIF Case Study Code

```
#include <iostream>
#include <string>
#include "GLeakage.h"

int main()
{
    std::cout << "Using LIBQIF Library Example" << std::endl;

    std::string channel_elements = "0.3 0.7; 0.7 0.3; 0.3 0.7";
    Channel C= Channel(channel_elements);

    std::string function_elements = "1 0 0; 0 1 0; 0 0 1";
    Gain g=Gain(function_elements);

    //Creating the probability distribution vectors
    std::string vector1_elements = "0.3333 0.3333 0.3334";
    std::string vector2_elements = "0 0.5 0.5";
    std::string vector3_elements = "0.5 0.5 0";
    std::string vector4_elements = "0.25 0.5 0.25";

    Prob p1= Prob(vector1_elements);
    Prob p2= Prob(vector2_elements);
    Prob p3= Prob(vector3_elements);
    Prob p4= Prob(vector4_elements);

    //GLeakage
    GLeakage gl= GLeakage(C,g);

    //calculating measures
    double Lg1=gl.leakage(p1);
    double Lg2=gl.leakage(p2);
    double Lg3=gl.leakage(p3);
    double Lg4=gl.leakage(p4);

    std::cout << "Lg p1" << Lg1 << std::endl;
    std::cout << "Lg p2" << Lg2 << std::endl;
    std::cout << "Lg p3" << Lg3 << std::endl;
    std::cout << "Lg p4" << Lg4 << std::endl;
}
```

Example 2 code.