Probabilistic Computation for Information Security

Piotr Mardziel
Department of Computer Science
University of Maryland
College Park, MD 20740
piotrm@cs.umd.edu

Kasturi something
Affiliation
Address
email

Probabilistic computation is a convenient means of mechanically reasoning about a variety of information security problems. At its core, information security concerns itself with measuring or limiting the knowledge and adversary might attain from interacting with a protected system. Probabilistic inference lets one compute this knowledge explicitly as long as the interaction can be described as a program in a suitable probabilistic language that supports conditioning. Security concerns, however, require soundness guarantees on probabilistic inference which are generally not present in machine learning applications. We summarize some recent work on probabilistic computing for information security and highlight challenging aspects that still need to be addressed.

Tracking knowledge Tracking adversary knowledge is a central tool for the applications to follow. We describe this core concept adopting the notation and conventions of Clarkson's experimental protocol [2].

Alice wishes to grant Bob partial access to her secret information while preserving the secrecy of some aspect of the secret. To do this, Alice will track what Bob learns about her secrets while assuming Bob is a rational adversary who will use any information provided to him in order to learn those secrets.

Alice's secret is a value s which Bob knows is in some set of possible secrets **Secrets**. Furthermore, Bob considers some secret values more likely than others, a probability distribution δ_1 : **Secrets** $\rightarrow$ $\mathbb{R}$. We will refer to this distribution as Bob's *belief*. Alice then considers the effect of letting Bob learn the output of a potentially non-deterministic function or *query* Q, written in a probabilistic language. She can do this by computing the conditional distribution $\delta_2 \stackrel{\text{def}}{=} \delta_1|(Q(s) = o)$ for some potential output value o. Assuming Bob learns nothing more than the output o and $\delta|C$ indeed implements *exact* probabilistic conditioning, this distribution δ_2 will be the belief Bob would attain, were he to learn that the function Q outputs o given secret input s.

If Bob indeed does learn this output, Alice can revise her representation of Bob's belief to δ_2 which she can then revise further if Bob learns additional functions over her secret data. In this manner Alice explicitly tracks what Bob knows about her secrets as probability distribution. The problems of how Alice knew δ_1, Bob's initial belief, in the first place and whether she can actually perform exact inference will need to be addressed.

Knowledge-based policy enforcement Alice can use the tracked belief in a variety of ways, the most direct of which is to judge the information security or safety of Q in order to determine whether she should let Bob get access to the output of the function. To do this, Alice starts by defining a knowledge-based policy P : **Dist** $\rightarrow$ $\{\mathsf{True}, \mathsf{False}\}$, where **Dist** is a set of all distributions over **Secrets**. For example, Alice might require that Bob never becomes too certain of the secret, so she defines $P(\delta) \stackrel{\text{def}}{=} \forall s \in$ **Secrets** . $\delta(v) \leq t$ for some threshold t. She will then deem the query Q safe whenever $P(\delta_2)$ holds. This particular policy is an application of Smith's vulnerability metric[6], which measures the expected likelihood of Bob guessing a secret value correctly in one try.

Unfortunately for Alice, exact probabilistic conditioning is expensive so she decides has to use an approximation. Instead of computing δ_2, she gets $\delta_2^{\sim} \stackrel{\text{def}}{=} \delta_1|^{\sim}(Q(s) = o) \approx \delta_1|(Q(s) = o)$. The approximation $\delta_2^{\sim}$ here need not be a proper probability distribution. She does not want the approx-

imate inference to conclude a query is safe when it really is not so she requires the approximation to be sound relative to her policy:

Definition. A probabilistic inference method $|\sim$ is *sound relative to a policy* P iff for every belief δ, if $P(\delta|C)$ fails then $P(\delta|\sim C)$ fails.

For Alice's example policy, a sound inference method would have to never underestimate the probability of any secret, though it could over-approximate it. In recent work[5], we demonstrate a probabilistic language capable of this approximate, but sound inference. Given the definition of conditional probability, $Pr(A|B) = Pr(A \wedge B)/Pr(B)$, the probabilistic interpreter maintains an over-approximation of $Pr(A \wedge B)$ and an under-approximation of $Pr(B)$, leading to a sound upper bound for $Pr(A|B)$. The work also shows how Alice can enforce her policy without revealing anything about her secret to Bob, by making her policy simulatable[3]. Further application of these ideas were proposed to protect information of multiple mutually distrusting parties participating in secure multi-party computation[4].

If Alice deems Q safe, she can allow Bob access to the output of Q and update her representation of Bob's belief to $\delta_2^{\sim}$. If she rejects Q, she will not allow access keep δ as the representation of Bob's unchanged belief. Whether Alice deems Q safe or not, she can repeat the same process for another program that might come around, starting from either δ or $\delta_2^{\sim}$, thereby maintaining an explicit representation of Bob's knowledge while enforcing her knowledge-based policy.

Obfuscation/Noising Probabilistic computation is also very convenient for discerning the effects of obfuscation or noising mechanisms, via simple composition with the observation function Q.

If Alice does not want Bob to learn the output of Q directly, she can add obfuscation to its input, or noising to its output. Let O be a potentially non-deterministic obfuscation function, written in a probabilistic language, that takes in a secret s and produces another s'. Depending on the structure of **Secrets**, this obfuscation might be in the form of resolution reduction, randomization of locations, or a variety of other mechanisms. Alice can then consider Bob's knowledge were he to learn the output of an obfuscated function $Q(O(s))$ instead of just $Q(s)$, using conditioning, $\delta_2 = \delta_1|\,(Q(O(s)) = o)$.

Alice can also consider adding noise to the output of Q, using a noising function N that takes in an output of Q and produces another, again written in a probabilistic language. She can then consider Bob's revised belief $\delta_2 = \delta_1|\,(N(Q(s)) = o)$. She might consider obfuscation and noise at the same time, revising Bob's belief to $\delta_2 = \delta_1|\,(N(Q(O(s))) = o)$. All of this assumes that Bob is aware of O and N.

Blacklist/Whitelist functions Having a means of determining the effects of obfuscation and noising allows one to design such functions to achieve security and utility goals. It is generally easy to create a noising function if only security is an issue. For example, Alice's example policy P above can be easily guaranteed by adding a noise function $N(s) \stackrel{\text{def}}{=} 42$, thereby ensuring Bob learns nothing from $N(Q(s))$. This ignores utility concerns that Bob (and Alice) might be interested in.

Alice and Bob can address this issue by agreeing to two functions, the blacklist function B and the whitelist function W. These are probabilistic functions that take in a secret as inputs and whose outputs Bob should not be able to predict well (blacklist) or should be able to predict well (whitelist). Alice's security concerns would be encoded in the blacklist function whereas Bob's utility concerns would be encoded as the whitelist function. For example, Alice might not want Bob to determine her religion from her secret information, but agrees that it is acceptable that Bob determines her age, regardless of whether these properties are directly part of the secret s.

In the framework of belief tracking, Alice can judge the security and utility inherent in Bob's belief δ by evaluating the blacklist and whitelist functions on belief δ : **Secrets** $\rightarrow \mathbb{R}$, written $B(\delta) = \delta_B$, a distribution on the outputs of B, and $W(\delta) = \delta_W$, a distribution on the outputs of W. This can be done using probabilistic languages by simply evaluating $B(s)$ or $W(s)$ where s is the probabilistic value tracking Bob's belief.

Alice can determine the safety of δ_B and utility of δ_W in a variety of ways. She could use the policy P from earlier on δ_B, bounding Bob's chances of guessing the output of B. She can also use the negation of P, specifically $\neg P(\delta) = \exists o\ .\ \delta(o) > t$, to measure some minimum level of

predictability of the whitelist function W, by checking that $\neg P(\delta_W)$ holds. Alternate means could involve the ubiquitous entropy quantity to measure at least a minimum entropy on δ_B and at most a maximum entropy on δ_W.

Definition. Blacklist/whitelist functions B and W are satisfied on belief δ relative to security policy P and utility policy U whenever $P(B(\delta))$ and $U(W(\delta))$ hold.

Having the ability to measure the secrecy and utility of Bob's belief, Alice can then determine which function Q to allow Bob to see the output of, so as to ensure $\delta|\,(Q(s) = o)$ satisfies security and utility policies. The direct choice is to let Bob see the output of the whitelist function W, assuming his knowledge from this observation does not let him attack Alice's blacklist function B. Otherwise Alice would need to find noising N and/or obfuscation functions O so that the output of $Q(s) = N(W(O(s)))$ is reasonable at ensuring utility, but at the same time reasonable at maintaining secrecy.

PM – Kasturi, can you say more about an example of how this is done in your paper and the open challenging problems from it. Make sure to cite it [1] and any other of your papers that might be relevant. Give a good advertisement for the work.

Predictive models for Blacklist functions The blacklist (and whitelist) functions are useful to protect information which is not part of the secret value s, but is somehow correlated with the secret value. For example, Alice's simple demographical information like age, postal code, and gender might compose the structure of s but Alice might be more interested in protecting her religions preference, even though this is not a field in the system she is securing.

Predictive models or classifiers, however, might exist that do a reasonable job of predicting religion given demographical information. Naturally, Alice could specify one such model as her blacklist function B. Unfortunately this does not stop Bob from using a different model B which is as good or better at predicting Alice's religion. This is especially problematic when there are two disjoint sets of fields in Alice's secret s that are both good for predicting religion. Alice might obfuscate one, but leave the second set completely untouched.

In general it is impossible for Alice to ensure *all* possible models for religion are defeated by her noising/obfuscation functions. This is simply due to Bob's potential model $\text{Religion}(s) = r$ where r is a constant that just happens to be Alice's religion. To make this problem reasonable, we need to restrict ourselves to models that Bob, assumed sane, could produce, given some body of data available to him.

Our goal is then, given a set of data instances D, with two class attributes, one "whitelist", and one "blacklist", to create noising/obfuscation functions that, when applied to the data instances, will make *all* classifiers for the blacklist attribute fail (perform poorly) but make some classifiers for the whitelist attribute perform well. PM – todo: Write this in a nicer way. Thinking about this now, it seems like it can all be described without need for probabilistic computing.

Initial distribution PM – todo: How to soundly determine what an adversary believes to begin with. If there is room left.

Related work

PM – todo: Add some other clear applications of probabilistic computing to other information security research, if there is room.

References

[1] Supriyo Chakraborty, Kasturi R. Raghavan, Mani B. Srivastava, Chatschik Bisdikian, and Lance M. Kaplan. Balancing value and risk in information sharing through obfuscation. In *Proceedings of the International Conerence on Information Fusion (FUSION)*, 2012.

[2] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[3] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. Simulatable auditing. In *Proceedings of the ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, 2005.

[4] Piotr Mardziel, Michael Hicks, Jonathan Katz, and Mudhakar Srivatsa. Knowledge-oriented secure multiparty computation. In *Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, 2012.

[5] Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa. Dynamic enforcement of knowledge-based security policies. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*, 2011.

[6] Geoffrey Smith. On the foundations of quantitative information flow. In *Proceedings of the Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, 2009.