The Phish-Market Protocol

Secure Sharing Between Competitors

The Phish-Market protocol encourages take-down companies to share information about malicious websites by compensating them for this data without revealing sensitive information to their competitors. Cryptography lets contributing firms verify payment amounts without learning which offered website URLs were "purchased."

Tal Moran and Tyler Moore
Harvard University

Consider the following situation: company A is a startup with a new product that it would like to market. Company B is a mailing-list provider and has a huge database of people's names, addresses, and attributes.

Based on its research, company A is convinced that people who like dogs and are more than six feet tall will respond favorably to an advertisement for its product. Thus, A would like to purchase a list of potential customers with these attributes from company B. However, A doesn't want to reveal its marketing strategy—if competitors discover that A's product is being targeted to tall dog-lovers, they might be able to scoop it.

One solution would be for company A to buy the entire database from company B. This would certainly hide any specific attributes A is looking for. However, this could be prohibitively expensive—the actual data A is interested in is only a tiny fraction of the database. Moreover, A might already have a list of potential customers compiled from other sources; A would prefer not to pay twice for data it already has. There might also be privacy or regulatory issues preventing B from selling too much of its information to one entity.

A second possibility would be for both companies to use a trusted third party (TTP). Company A can disclose its secret strategy and the list of names it already knows to the TTP while company B discloses its entire database. In this case, the TTP could search the database for tall dog-lovers who aren't on A's "already known" list, send those to A, and reveal only the total sum company A must pay to company B in return for this information.

Unfortunately, in practice, two companies often have trouble finding a third party that they both completely trust. Situations like these are where modern cryptography can truly shine. Using a cryptographic protocol, companies A and B can emulate a *virtual trusted party* that provides the same services a real one would, but without having to trust in anything but their own implementation of the protocol.

We've constructed an efficient protocol that solves this problem, which we believe can be applied to many similar data-sharing problems.

Phishing Website Take Down

Our original motivation came from a seemingly unrelated real-life problem: an attempt to encourage data sharing between phishing take-down companies.

Phishing is the criminal activity of enticing people to visit websites that impersonate genuine bank websites and dupe visitors into revealing passwords and other credentials. (Although a wide range of companies have been subject to phishing attacks, most are financial institutions; for simplicity, we use the term "banks" for firms being attacked.) One key countermeasure to phishing is to promptly remove the imitation websites. Banks can achieve removal by erasing the webpages from the hosting machine or contacting a registrar to suspend a domain name from the Domain Name System (DNS), so the fraudulent host can no longer be resolved.

Although some banks deal with phishing website removal exclusively in-house, most hire specialist take-down companies to carry out the task. Such com-

Table 1. Timed and financial exposure to phishing attacks caused by not sharing data.*

Exposure figures (six-month totals)	τ_A's client banks	τ_B's client banks
Actual values	1,005,000 hrs ($276 million)	78,000 hrs ($32.0 million)
Effect of not sharing	587,000 hrs ($163 million)	17,000 hrs ($3.5 million)
Expected exposure if sharing	418,000 hrs ($113 million)	61,000 hrs ($28.5 million)

*Numbers in red indicate the portion of phishing risk caused by slower take down.

panies—typically, divisions of brand-protection firms or information security service providers—perform two key services. First, they're good at getting phishing websites removed quickly, having developed relationships with ISPs and registrars across the globe and deployed multilingual teams at 24/7 operation centers. Second, they collect a more timely and comprehensive listing of phishing URLs than banks normally gather.

Most take-down companies view their URL feeds as a key competitive advantage over banks and other take-down providers. However, recent work has shown that the feeds such companies compile suffer from large gaps in coverage that significantly prolong the time needed to remove phishing websites. Tyler Moore and Richard Clayton examined six months of aggregated URL feeds from many sources, including two major take-down companies.[1] They found that up to 40 percent of phishing websites remained unknown to the company with the take-down contract but were discovered by others. Another 29 percent were discovered by the responsible take-down company only after others had identified them. By measuring these missed websites' substantially longer lifetimes, Moore and Clayton estimated that 120 banks served by these two companies collectively risk at least US$330 million per year by failing to share proprietary URL feeds (see Table 1).

Figure 1 gives the details for one take-down company, labeled τ_A. The left circle represents the phishing sites that appear in τ_A's feed, and the right circle the sites that appear in its competitors' feeds. The intersection contains the sites that appear in all feeds, the top half representing sites that τ_A found before other companies, and the bottom those others found before τ_A.

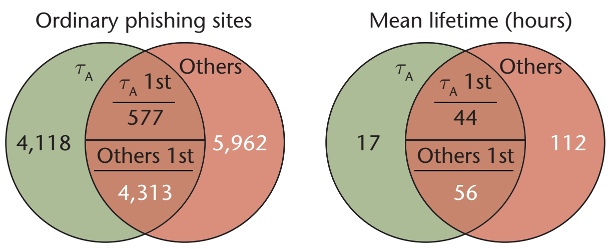

Figure 1. Missed phishing websites. We can see how substantial incompleteness in one take-down company's URL feed slowed removal of phishing websites impersonating 54 banks.[1]

The Phish-Market Protocol

Named after its applicability to sharing phishing URLs, the Phish-Market protocol can efficiently solve data-sharing problems while overcoming the sharing entities' competitive concerns. Contributors are compensated, but sensitive details that might be inferred from a transaction are hidden from both parties, all without requiring access to a TTP. The protocol aims to let a *buyer* acquire new records, such as phishing URLs or customer details, from a *seller*. The buyer is interested only in a subset of data matching one or more *tags*, attributes describing the records. Tags can indicate which bank is the subject of the phishing URL or customer attributes in a database.

At a high level, the Phish-Market protocol does the following:

- shares only those records that match the tags the buyer requests;
- hides from the seller which records the buyer receives;
- hides from the seller which tags the buyer requests; and
- securely tallies the number of records the buyer receives without double-counting records from the seller that the buyer already has.

These requirements are essential for take-down companies considering sharing phishing URL data. The first requirement ensures that only URLs pertaining to banks that the take-down company is interested in are exchanged. The second and third requirements protect the acquiring firm from revealing too much about its business posture, such as its client banks and its weaknesses in phishing-website-detection capability. The fourth requirement assures both the buyer and seller that the exchange is fair. We anticipate that take-down companies would execute the protocol as both buyers and sellers, and that only the net contributor would be compensated financially.

The requirements also match up nicely to the customer database provider problem. In fact, we envision many data-sharing applications that could benefit from a sharing mechanism like the Phish-Market protocol. For instance, hospitals might be willing to share de-identified patient records with medical researchers. Instead of sharing entire databases, they might share

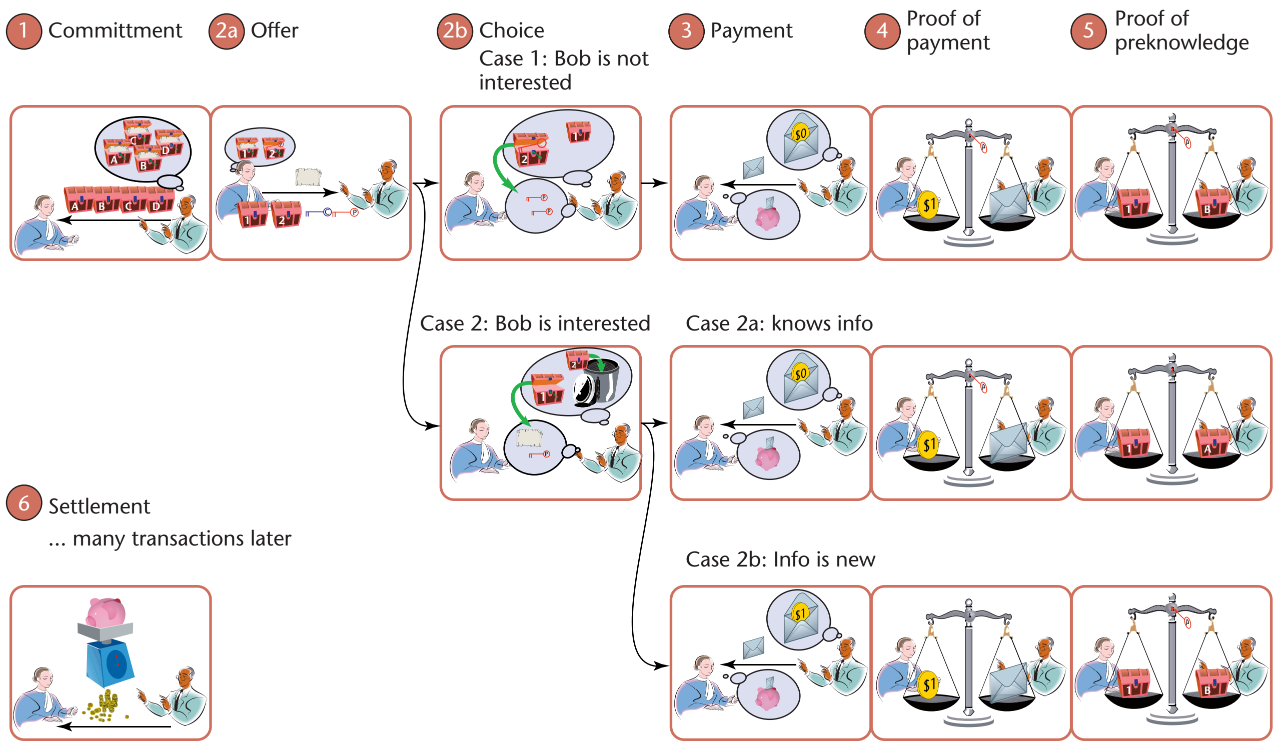

Figure 2. Overview of the Phish-Market protocol. Columns correspond to protocol steps, whereas rows correspond to the different situations that might occur during execution (note that to the seller, the different situations are indistinguishable).

only those records matching a combination of diseases and other selected attributes. Oil companies might like to purchase geographical data on seabeds without revealing their exploration plans to competitors.

Other types of information security data lists, such as DNS and IP blacklists, could also be more effective if shared more widely among security firms.

Protocol Overview

We now describe the Phish-Market protocol at a high level, keeping the cryptography details to a minimum. A detailed technical description will appear in a companion paper.[2] We use several cryptographic primitives—*commitments*, *zero-knowledge proofs*, and *oblivious transfer*—which we describe briefly; readers can find formal definitions elsewhere.[3,4]

The entire protocol takes six steps to complete, as Figure 2 illustrates, with Bob acting as buyer and Sally as seller.

Step 1: Buyer Commitment

Bob first commits to his current state of knowledge—all the records in his database—and sends his commitment to Sally. Cryptographic commitments bind Bob to his state of knowledge while keeping the information hidden from Sally. This commitment will let Bob withhold payment for records he already knows.

Step 2: Seller Offer and Buyer Choice

Sally sends Bob an offer for each record she wishes to sell. Included in the offer is a tag, a commitment to the record, and a "proof key" (explained in steps 4 and 5). Bob can use the tag to decide whether he's interested in the record Sally is offering. If Bob chooses not to learn the record, he will instead learn a second proof key that will let him withhold payment for this record later in the protocol (without revealing the fact to Sally).

To keep Bob's choice secret from Sally, Bob and Sally execute an Oblivious Transfer (OT) protocol. We can think of this OT protocol as Sally placing the record and a proof key in two locked boxes. The boxes are labeled and given to Bob, along with a key that can open either box (but that Bob can only use once). If Bob decides he's interested in the record, he opens the box with the record; if not, he opens the other box to get the second proof key.

Step 3: Payment

Bob must now pay up for the new record learned from Sally. Of course, Bob should pay Sally only if he chooses to learn the record and doesn't already know it. Hence, he can pay using a *real* or *fake* payment. A real payment is a cryptographic commitment to the number 1, whereas a fake payment is a commitment to the number 0. In this context, a commitment from

Bob is a public-key encryption of a value (0 or 1) using a key known only to Bob. Sally can't infer the commitment's value without the key, but Bob can potentially open the commitment later and thereby prove to Sally the true value committed. In fact, Bob will never open the payment commitment because this will reveal too much information to Sally. Instead, we use commitments with a special *homomorphic* property that will let Bob prove the total number of real payments made in the settlement phase (step 6) without opening any specific payment.

Step 4: Proof of Payment

After sending the committed payment to Sally, Bob must demonstrate that it is legitimate. Of course, the payment is only real if Bob committed to 1 and not 0. This is where the proof keys come in: using a proof key, Bob can fake a proof and convince Sally that a payment is real even if it isn't. Technically, Bob uses a zero-knowledge proof to convince Sally that either the payment is real or he uses a proof key (a zero-knowledge proof lets the prover convince the verifier of a statement without revealing anything except for the statement's validity). Figure 2 represents the proof with a balance scale; the difference between real and fake coins is reflected in their weight (a fake coin, for example, weighs nothing, whereas a real coin weighs 1 gram). Bob places the envelope with his payment on one side of the balance and a real coin on the other. If Bob's payment is real, Sally will see that the balance shows the two are equivalent. If Bob doesn't actually pay up, he can use the proof key to "lock" the balance into place and make the different weights appear equal.

Step 5: Proof of Prior Knowledge

Bob must give Sally one more zero-knowledge proof, this time regarding whether he knew about the record before Sally gave it to him. If Bob chose not to learn the record in step 2, he can't possibly prove that he already knew it. Alternatively, Bob might have identified a previously unseen record from Sally. In both cases, Bob uses a proof key to generate a fake proof claiming prior knowledge of the record.

The protocol design restricts how many proof keys Bob has to ensure that he makes a real payment if he learns a new record. If Bob had chosen to learn the record in step 2, he'd be left with a single proof key. With this remaining key, he can either fake the proof of payment in step 4 or the proof of prior knowledge in step 5, but not both. If he chooses to use the proof key to prove prior knowledge of the record (as he's forced to do if the record is new information), he must provide a real proof of payment in step 4.

Step 6: Settlement

After running steps 1 through 5 many times to exchange unknown records, Bob and Sally can finally settle up. The payment commitments' homomorphic property lets anyone take two commitments and, without opening either one, compute a commitment to the sum of their values. Figure 2 represents this as weighing the piggy bank containing the payments: we don't need to open it up to verify the total weight (and from the total weight, no one can tell which individual payments are real and which are fake).

Using homomorphic addition, Sally can tally up the sum of the many 1 and 0 payment commitments used in step 3, even though she can't open them. Sally takes the payments from multiple executions and computes a commitment to the total payment, which represents the number of records actually "sold." Bob can then open Sally's aggregated commitment. Sally learns only the total number of real payments received (and not which individual payments were real). Bob and Sally can use this sum as the basis for a monetary transaction.

Security Discussion

Let's now look at the security guarantees made separately for the seller and buyer. For the buyer, we guarantee two properties: first, the seller doesn't learn anything about the set of tags the buyer is interested in; second, the seller doesn't learn anything about the set of previously known records. The only exception to these guarantees is what information the seller can deduce from the payment amount.

The seller's primary interest is to ensure that she's being justly compensated for each record that the buyer learns from her. We guarantee that the seller's security in the distributed protocol is the same as for an "ideal" protocol involving a TTP, with two caveats: first, the buyer will learn the tags of all the seller's records (given that those are sent in the clear), rather than just those of the records he pays for (as would be the case using an ideal trusted party). Second, the buyer learns which records he already knows that are also in the seller's database (an ideal third party would send the buyer only records he doesn't already know). The reason for both relaxations is efficiency. In particular, letting the buyer learn records he already knows (albeit without paying for them) lets him perform the database lookup himself rather than engage in a large, joint secure computation with the seller.

Note that some attacks would still be possible even if a TTP were available. For example, no guarantee exists that the records sold will be useful or correctly tagged. A malicious seller could send random strings instead of records, forcing the buyer to pay for garbage records (because they wouldn't appear in the buyer's database). A malicious seller could also attack the buyer's privacy: if she uses the same tag for all the records in a certain period, she can tell whether the buyer is interested in the tag if he made a payment at the end of that period.

Fortunately, mitigating strategies are typically available to counter these threats when using the protocol in the real world. First, the buyer can evaluate the records he learns and set the price he's willing to pay for each one based on the quality of records he received in the past. If he determines that the seller is providing low-quality records, the buyer can request a lower dollar price per record or refuse to do business with that seller in the future. This would mitigate the garbage record attack. Defending against the privacy breach attack is harder—the payment will always leak some information about which tags the buyer is interested in. We can help the buyer detect such attacks by compromising a little on the seller's privacy: if we give the buyer all the tags the seller uses (without the corresponding records), the buyer can verify that no set of tags is overly represented.

Finally, in a two-party protocol, unlike a protocol that uses a TTP, each side can decide to abort the protocol prematurely. This affects our protocol's security if the buyer decides to abort after learning a record but before making the payment. However, the same problem exists in many remote transactions (when purchasing physical goods over the phone, for instance, a seller can refuse to send the goods after receiving payment). Sellers can use the same legal frameworks to handle a refusal to pay in this case.

Protocol Efficiency

Powerful results from theoretical cryptography demonstrate how we can convert any task using a TTP into one that doesn't require third parties.[5,6] However, these techniques are usually inefficient. The most efficient implementations of general techniques (such as the Fairplay system[7]) are still orders of magnitude too slow for practical use on the scale required to share large numbers of records. Even relatively simple computations, such as those needed to hold sugar beet auctions in Denmark,[8] require specialized protocols to overcome this problem.

Fast operation is critical when it comes to distributing phishing URL feeds—the longer a phishing website remains online, the more customer credentials might be at risk. Fortunately, the Phish-Market protocol is much more efficient than generic secure computation.

We implemented an elliptic-curve- (EC-) based version of the protocol in Java, using the Qilin Crypto SDK (software developer's kit)[9] for the high-level cryptographic primitives and the Bouncy Castle Crypto API (www.bouncycastle.org) for low-level EC operations (full code for the implementation of our protocol is available elsewhere[9]).

We simulated both sides of the protocol on a single server with one dual-core 2.4-GHz Intel Xeon processor and 2 Gbytes of memory. (The main bottleneck in the protocol is the CPU—one transaction requires less than 3 Kbytes of communication—so running both sides on one server would only cause us to overestimate the running time.)

To test the protocol's performance under real-world conditions, we used the phishing URL feeds from two large take-down companies during the first two weeks of April 2009. We assigned one take-down company to be the seller and the other to be the buyer (we ran experiments with both companies playing both roles). For the two-week sample period, one company found 8,582 unique URLs, whereas the other discovered 17,721. The first company was interested in obtaining phishing URLs for 59 banks, and the second for 54 banks, according to the client lists shared with us.

The primary metric we use to measure our implementation's performance is the time required to process and transmit each phishing URL from the seller to the buyer. The less time required, the closer the URL sharing is to instantaneous. On average, each URL faced a very acceptable delay of 5.13 seconds to complete the exchange (3.19-second median). Two main factors affect the total delay. First is the processing time required to execute the protocol. This computational time was very consistent, taking an average of 2.37 seconds but never more than 4.02 seconds. The other, less predictable, reason for delay happens whenever many phishing URLs are discovered around the same time. Whenever a clump of URLs were reported, some URLs had to wait for others to be processed, leading to a longer delay. Although a multithreaded implementation could minimize these queue delays (by utilizing more CPU cores), we chose to implement the protocol using a single thread to demonstrate its feasibility even with modest hardware. Moreover, note that we optimized the protocol implementation for clarity and source code generality rather than speed. The average queue delay due to waiting on other URLs to finish processing was 2.76 seconds, and the longest delay was 34.6 seconds.

To give readers a better feel for how the processing time varies, Figure 3 plots the cumulative distribution functions for the time taken to process each URL, the time that URL spent waiting in the seller's queue, and the total delay between the time the URL entered the seller's queue and the time the buyer received it. The computer processed 48.4 percent of the URLs in less than 3 seconds, yet 9.7 percent took more than 10 seconds. Despite the variation, no URL took more than 37 seconds to process. Given that phishing website removal requires human intervention, a 37-second delay is negligible, and certainly much better than the many days longer unknown sites currently take for removal!

In addition to the total delay (red dash-dot line), Figure 3 plots the delay's two key components. The green dashed line appears nearly vertical around 2 seconds, suggesting that the per-URL processing time is very consistent. Meanwhile, the blue solid line plots the queue delay, which accounts for the overall delay's stretched tail. Hence, if the queue delay were reduced via multiple processors or threads, the total delay might approach the consistently shorter processing time.

In exchange for these very modest delays, takedown companies can trade information on new phishing websites so that the overall lifetime of such sites is reduced as much as half,[1] while crediting the contributing firm.

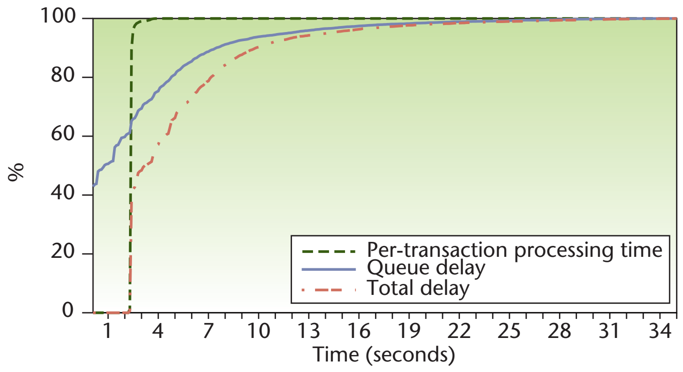

Figure 3. Observed cumulative distribution function of the time required to share each phishing URL. The measured delay is between the time the seller first learned the URL and the time it becomes known to the buyer. The red dash-dot line denotes the total real-time delay, and the others denote subcomponents of the delay.

Balancing the advantages of data sharing with the risks of helping competitors or experiencing privacy breaches is hard. Yet markets, both legitimate and illicit, are becoming increasingly data-driven. From identifying users for targeted advertising to blocking spam and shutting down phishing websites, sharing data is now essential because no single company has a complete view of the global environment.

We've designed and implemented a practical mechanism for competing, untrusting companies to selectively buy and sell the information that's relevant to their needs without leaking too much additional data. Although our implementation is focused on the specific data-sharing problem that phishing takedown companies encounter, we can directly apply it to several other problems (such as the mailing-list provider/targeted advertiser example in the introduction) and employ our techniques in many similar situations.

In fact, many data-sharing problems have more relaxed requirements than those the Phish-Market protocol must satisfy (for instance, an exploring oil company won't buy data about locations for which it already has information, so the requirement that the company not pay for data already known isn't needed). In these cases, our protocol can be made simpler and even more efficient.

A key lesson from our experience is that modern cryptographic tools can be very helpful in resolving the tension between sharing and secrecy. The techniques are now efficient enough to be practical and allow an unprecedented level of control over what companies can reveal and what they must not. In today's data-driven economy, these tools should be found in every developer's toolbox. ☐

Acknowledgments

We thank Allan Friedman for suggesting the mailing-list problem described in the introduction.

References

1. T. Moore and R. Clayton, "The Consequence of Noncooperation in the Fight against Phishing," *Proc. Anti-Phishing Working Group eCrime Researchers Summit* (APWG eCrime 08), IEEE Press, 2008, pp. 1–14.
2. T. Moran and T. Moore, "The Phish-Market Protocol: Securely Sharing Attack Data between Competitors," *Financial Cryptography and Data Security*, LNCS 6052, R. Sion, ed., Springer, 2010, pp. 222–237.
3. O. Goldreich, *Foundations of Cryptography: Basic Tools*, vol. 1, Cambridge Univ. Press, 2001.
4. O. Goldreich, *Foundations of Cryptography: Basic Applications*, vol. 2, Cambridge Univ. Press, 2004.
5. O. Goldreich, S Micali, and A. Wigderson, "How to Play Any Mental Game or a Completeness Theorem for Protocols with Honest Majority," *Proc. 19th Ann. ACM Symp. Theory of Computing* (STOC 87), ACM Press, 1987, pp. 218–229.
6. M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," *Proc. 20th Ann. ACM Symp. Theory of Computing* (STOC 88), ACM Press, 1988, pp. 1–10 (extended abstract).
7. D. Malkhi et al., "Fairplay—A Secure Two-Party Computation System," *Usenix Security Symp.*, Usenix Assoc., 2004, pp. 287–302.
8. P. Bogetoft et al., "Secure Multiparty Computation Goes Live," *Financial Cryptography and Data Security*, LNCS 5628, R. Dingledine and P. Golle, eds., Springer, 2009, pp. 325–343.
9. T. Moran, "The Qilin Project: A Java SDK for Rapid Prototyping of Cryptographic Protocols," 2009, http://qilin.seas.harvard.edu.

***Tal Moran** is a postdoctoral fellow at the Center for Research on Computation and Society at Harvard University. Contact him at talm@seas.harvard.edu.*

***Tyler Moore** is a postdoctoral fellow at the Center for Research on Computation and Society at Harvard University. Contact him at tmoore@seas.harvard.edu.*