The Phish Market Protocol: Securely Sharing Attack Data Between Competitors

Tal Moran and Tyler Moore

Center for Research on Computation & Society, Harvard University
{talm},{tmoore}@seas.harvard.edu

Abstract. A key way in which banks mitigate the effects of phishing is to remove fraudulent websites or suspend abusive domain names. This 'take-down' is often subcontracted to specialist firms. Prior work has shown that these take-down companies refuse to share 'feeds' of phishing website URLs with each other, and consequently, many phishing websites are not removed because the firm with the take-down contract remains unaware of their existence. The take-down companies are reticent to exchange feeds, fearing that competitors with less comprehensive lists might 'free-ride' off their efforts by not investing resources to find new websites, as well as use the feeds to poach clients. In this paper, we propose the Phish Market protocol, which enables companies with less comprehensive feeds to learn about websites impersonating their own clients that are held by other firms. The protocol is designed so that the contributing firm is compensated only for those websites affecting its competitor's clients and only those previously unknown to the receiving firm. Crucially, the protocol does not reveal to the contributing firm which URLs are needed by the receiver, as this is viewed as sensitive information by take-down firms. Using complete lists of phishing URLs obtained from two large take-down companies, our elliptic-curve-based implementation added a negligible average 5 second delay to securely share URLs.

1 Introduction

Phishing is the criminal activity of enticing people into visiting websites that impersonate genuine bank[1] websites, and to dupe them into revealing passwords and other credentials to carry out fraudulent activities. One of the key countermeasures to phishing is the prompt removal of the imitation bank websites. Removal may be achieved by erasing the web pages from the hosting machine, or by contacting a registrar to suspend a domain name from the 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. Take-down companies – typically divisions of brand-protection firms or information security service providers – perform two key services for banks. First, they are good at getting phishing websites removed quickly, having developed relationships with ISPs and registrars across the globe and deployed multi-lingual teams at 24x7 operations centers. Second, they collect a more timely and comprehensive listing of phishing URLs than banks normally gather.

[1] Although a wide range of companies have been subject to phishing attacks, the vast majority are financial institutions; for simplicity, we use the term 'banks' for firms being attacked.

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 compiled by take-down companies suffer from large gaps in coverage that significantly prolong the time taken to remove phishing websites. Moore and Clayton examined six months of aggregated URL feeds from many sources, including two major take-down companies [13]. They found that up to 40% of the phishing websites impersonating banks hired by take-down companies were known to others but not by the company with the take-down contract. Another 29% of websites were discovered by the responsible take-down company only after others had identified the sites. By measuring the substantially longer lifetimes of these missed websites, Moore and Clayton estimated that at least $330 million per year is being put at risk by the failure to share proprietary feeds of URLs for just the two companies they studied.

But is sharing the answer, and, if so, then how should an effective sharing mechanism be designed? Moore and Clayton appealed to the security industry's sense of responsibility and argued that URL feeds should be shared freely between take-down companies and banks, pointing to the precedent of sharing in the anti-virus industry. However, there are some reasonable objections to a sharing free-for-all. First, competition between take-down companies may drive investment into better techniques for identifying new phishing websites faster, and mandated sharing might undermine the incentive to innovate. Unsurprisingly, most take-down companies would rather see banks purchase the services of several take-down providers to overcome gaps in coverage.

In this paper, we describe the Phish Market protocol, which addresses the competitive concerns of take-down companies so that widespread sharing can take place. To bolster the incentive to share, our protocol enables sharing of URLs where the net contributors are compensated without revealing the sensitive details of what is shared to competitors. At a high level, the Phish Market protocol does the following:

1. shares only those URLs that the receiving party wants (i.e., the banks the receiving party works for);
2. does not reveal to the providing party which URLs are given to the receiving party;
3. securely tallies the number of URLs given to the receiving party;
4. does not count URLs the receiving party already has.

Timing is critical when it comes to distributing URL feeds — the longer a phishing website remains online, the more customer credentials may be at risk. While in theory generic multiparty computation protocols can be used to implement this mechanism, in practice they are very inefficient and would introduce significant delays in processing the many thousands of phishing websites. In contrast, our custom protocol is extremely efficient (and still provably secure).

To demonstrate the feasibility of our mechanism, we have implemented an elliptic-curve-based version of the protocol in Java. Using the feeds from two take-down companies during the first two weeks of April 2009, we tested protocol performance in a real-world scenario. We found that our sharing protocol introduces an average delay of 5 seconds to the processing and transmission per phishing URL. In exchange for this very short delay, information on new phishing websites is exchanged between take-down companies so that the overall lifetime of phishing websites may be halved [13] while crediting the contributing firm.

2 The Phish Market Protocol

We now describe the Phish Market protocol, where companies with more comprehensive feeds of phishing URLs are compensated for sharing with those who learn most from sharing. The protocol deals with a number of constraints in order to satisfy the exchanging parties without relying on a trusted third party. While we formalize the security properties guaranteed in Section 2.2, it is helpful to first mention the requirements affecting the protocol's design. In particular, each company is only interested in a subset of their competitors' feeds, namely those URLs that affect their own customers. As an added complexity, take-down companies keep their list of client banks secret from competitors. Hence, we need a way to share only those URLs that the other party is interested in, without revealing which URLs are being shared. Note that our mechanism does not directly compensate contributors; instead, it tallies the total number of useful URLs exchanged in a way that cannot be manipulated by either party.

An Optimal Ideal-World Protocol. We describe the task our protocol performs by first explaining how it could be done if we used a trusted third party (TTP) — someone who was entirely trusted by both the contributor (or *Seller*) and the receiver (or *Buyer*). To share data in this ideal scenario, both the Buyer and the Seller would send the data to the TTP; the Buyer's data consists of the URLs she already knows and her list of client banks, while the Seller's data consists of the URLs he is attempting to sell and their classification (i.e., which bank each URL is attempting to impersonate). The TTP could then send the Buyer only those URLs that both impersonate her clients and that she did not already know. The TTP would send the Seller the number of URLs sent to the Buyer. This number would then be used to compute the compensation owed to the Seller. Since the TTP only sends the new "interesting" URLs to the Buyer, she will not learn anything about URLs she was not interested in (and would not have to pay for them). On the other hand, the TTP sends the Seller only the number of URLs sold, not the URLs themselves. Consequently, the Seller will not gain additional information about the Buyer's client list.

Our protocol is intended to provide this functionality, maintaining its privacy properties, but without requiring a third party. Using powerful results from theoretical cryptography, it is known how to convert any task that can be performed with the aid of a TTP to one that does not require third parties. However, these techniques are usually inefficient. In our case, even the most efficient implementations of general techniques (such as the Fairplay system [11]) would be orders of magnitude too slow for practical use.

We give an efficient protocol for executing a single 'transaction' of the following form: the Seller first sends a 'tag' to the Buyer. The tag can be, for example, the name of the bank associated with the URL to be sold. The Buyer uses the tag to decide whether or not she is interested in learning the corresponding URL. She also commits in advance to the set of URLs she already knows. If the Buyer was interested in the tag and did not already know the corresponding URL, the Seller receives a 'payment'. Otherwise, the Seller receives a 'counterfeit payment' (the Seller should not be able to tell whether or not a payment is counterfeit — otherwise he would be able to tell whether or not the Buyer was interested in the URL, and thus discover the Buyer's client list).

At the end of some previously agreed period (or number of transactions), the Buyer reveals to the Seller how many 'real' payments were sent, and proves that this is indeed the case (without revealing which of the payments were real). In practice, we envision each pair of take-down companies executing the basic protocol in both directions: when one of the companies acquires a new URL, it would execute the protocol as the Seller, with the other company playing the Buyer. When the second company acquires a new URL, it would execute an instance of the protocol in the other direction, with the first party as Buyer and the second as Seller.

Note that, even in the using a trusted third party, some attacks are still possible. For example, there is no guarantee that the URLs sold will be useful or correctly tagged. A malicious Seller could send random strings instead of URLs, forcing the Buyer to 'pay' for garbage URLs (since they would not appear in the Buyer's database). A malicious Seller can also attack the Buyer's privacy: if he uses the same tag for all the URLs in a certain period, the Seller can tell whether or not the Buyer is interested in the tag by whether or not a payment was made at the end of the period.

Since these attacks can be carried out in the ideal world, any protocol implementing this type of exchange is also vulnerable. For the situations in which we anticipate our protocol will be used, however, there are mitigating strategies. First, the Buyer can evaluate the URLs she learns and set the price she is willing to pay for each URL based on the quality of URLs she received in the past. If she determines that the Seller is providing low-quality URLs, the Buyer can request a lower dollar price per URL or refuse to do business with that Seller in the future. This would mitigate the "garbage URL" 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 this type of attack by compromising a little on the Seller's privacy: if we give the Buyer all the tags the Seller uses (without the corresponding URLs), the Buyer can verify that no set of tags is overly represented.

Finally, in a two-party protocol, unlike a protocol that uses a trusted third party, each side can decide to abort the protocol prematurely. This affects the security of our protocol if the Buyer decides to abort after learning a URL but before making the payment. However, the same problem exists in many remote transactions (e.g., when purchasing physical goods over the phone, the seller can refuse to send the goods after receiving payment). The same legal frameworks can be used to handle a refusal to pay in this case.

Below, we describe the protocol as well as the precise security guarantees we make.

2.1 Protocol overview

Payment Commitments. Before we describe the protocol itself, we must clarify what we mean by 'real' and 'counterfeit' payments. Our protocol uses cryptographic commitments as payment tokens. Loosely speaking, a commitment to a value x can be thought of as a public-key encryption of x, for which only the Buyer knows the secret key; the Seller can't tell what x is from the commitment, but the Buyer can 'open' a commitment and prove to the Seller that the commitment is to a specific value. In our protocol, a 'real' payment is a cryptographic commitment to the number 1, while a 'counterfeit' payment is a commitment to the number 0.

The payment commitments used by the protocol have a special property that allows them to be efficiently aggregated, even in encrypted form (they are *homomorphic*; see App. B for a formal definition). Thus, the Seller can take the 'payments' from multiple executions of the basic protocol and compute a commitment to the total payment (the number of URLs actually 'sold').

The Buyer will eventually open the aggregated commitment. At this point, the Seller will learn only the total number number of 'real' payments received (and not which individual payments were real). This value can be used as the basis for a monetary transaction between the two parties.

Protocol Construction. One of the more difficult challenges to solve efficiently is that the Buyer should not have to pay for URLs she already knew, while simultaneously protecting the privacy of the Buyer's client list. The known techniques for general secure computation of a function require an expensive public-key operation for each input (or even each bit of the input). In our case, the input would have to include the set of previously known URLs, which may be very large: A typical take-down company could learn an excess of 10 000 URLs per month, making existing systems impractical.

To solve this problem, we let the Buyer perform the database search locally, after learning the URL. If she discovers the URL in the database, she must then prove to the Seller that the URL existed in the database before the start of the transaction. However, this proof cannot use the URL itself, since that would reveal to the Seller that the Buyer was interested in it (thus exposing one of the Buyer's clients). The main idea behind the protocol is to split the proof into two:

1. The first proof is a 'proof of payment'. The payment in this case is a commitment to the value 1; the proof of payment proves that the Buyer can open the commitment she sent to the value 1.
2. The second proof is a 'proof of previous knowledge'. This proof convinces the Seller that the Buyer knew the URL before the start of the protocol.

The essence of the protocol is that we allow the Buyer to 'fake' a proof if she knows a corresponding secret key. The protocol is set up so that the Buyer initially knows a single secret key: she can fake the first proof or the second proof, but not both. Once the Buyer learns the tag, she must make a choice: she can either learn the corresponding URL, or learn the second secret key (but not both). Thus, if she chooses not to learn the URL, the Buyer can send a counterfeit payment (a commitment to 0), and fake both proofs. If she chooses to learn the URL and did not already know it, she is forced to fake the second proof, and therefore cannot fake the first (so she must send a real payment). The proofs we use are *Zero-Knowledge* (ZK) proofs: the Seller learns nothing from the proof except the validity of its statement. This protects the security of the Buyer (the Seller cannot tell whether or not the Buyer was interested in the URL or whether she previously knew it).

Fig. 1 shows a graphical overview of the protocol. We split the second proof into the boxes labeled *ZK Proof #2* and *Proof #3* in the figure. Before the protocol begins, the Buyer sends the Seller a commitment to her set of previously known URLs (see App. B for a more in-depth explanation of set commitments). ZK Proof #2 proves the Buyer holds a commitment for the URL (this part can be faked using a secret key).

Proof #3 proves the Buyer knew the commitment before the protocol began (this part cannot be faked; however, if the Buyer faked ZK Proof #2 she can choose an arbitrary commitment and prove she knew that). The reason for the split is that Proof #3 can be performed very efficiently, while Proof #2 requires public-key type operations. The numbers on the left and right-hand sides of the figure reference the corresponding lines in the full protocol listing (on the left these are the lines in Prot. 1a, and on the right in Protocols 1b and 2).

To simplify the presentation, the protocol in Fig. 1 omits two steps present in the full protocol:

1. The Buyer must prove that the payment is valid (either a commitment to 0 or a commitment to 1). Otherwise, if the Buyer fakes the first proof she could send a commitment to a negative number instead of a zero commitment (in which case the aggregate commitment would be opened to a lower value than the actual payment due).
2. The use of a Merkle tree as a set commitment (Proof #3) is not completely secure if the same Merkle tree is used for multiple transactions. This is because every execution of the protocol requires the Buyer to reveal a path from some leaf in the tree to the root. If the Seller sees the same leaf twice, he will learn that in at least one of the transactions the Buyer was using a "fake". To prevent this attack, the Buyer must make sure the tree also contains "chaff" commitments. When a fake commitment is needed, the Buyer uses one of the chaff commitments. The Buyer makes sure to use each chaff commitment at most once (she can add chaff commitments to the tree if they run out).

2.2 Security Properties

Unlike errors in most computer algorithms, protocols with faulty security may perform flawlessly — often by definition a failure in security is one that is undetected. Thus, an important part of the specification for any secure protocol is a formal definition of its security properties and an analysis of the conditions under which they are guaranteed.

We make separate security guarantees for the Seller and for the Buyer in each transaction (execution of the basic protocol).

Buyer's Security. The Buyer in our protocol has as input a set of tags in which she is interested, T (e.g., the list of banks she has as clients) and a set of previously known URLs, U. The security guarantee for the Buyer is that a malicious Seller does not learn anything about T or U, beyond what he can deduce from the payment amount. This is important because competitors naturally do not wish to reveal weaknesses (in terms of gaps in URL coverage). On the other hand, the Seller does not want to reveal URLs to the Buyer that the Buyer is unaware of without compensation. Finally, the Buyer does not want to reveal its client list to the Seller.

More formally:

Theorem 1 (Buyer's Security). *For any two sets of inputs* (T_0, U_0) *and* (T_1, U_1)*, such that* $|U_0| = |U_1|$*, the Seller's view of a protocol execution when the Buyer is given input* (T_0, U_0) *is statistically indistinguishable from its view when the Buyer is given input* (T_1, U_1)*.*

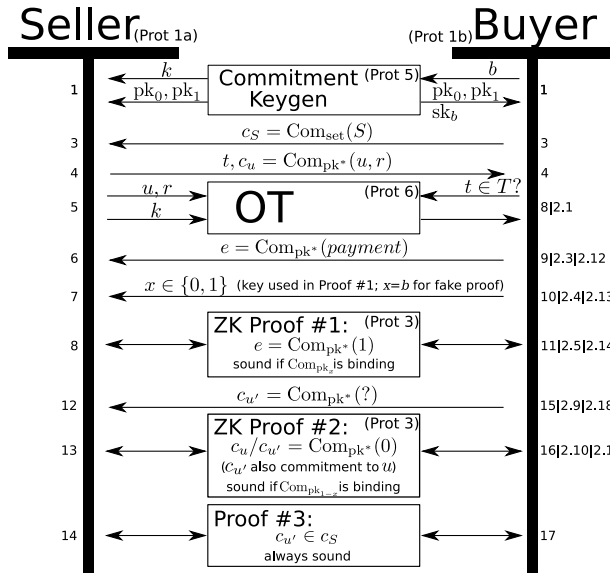

Fig. 1. Simplified Phish Market protocol overview.

Note that the Seller's view of the protocol does *not* include the opening of the aggregate payment: the Seller will obviously gain some information about T and U from the payment amount — what the theorem implies is that this is *all* the Seller learns. The intuitions behind the proof of this theorem appear in Appendix A.1

Seller's Security. Essentially, the Seller needs to ensure that he is being justly compensated for each URL that the Buyer learns from him. We define the security of the Seller by formally comparing our protocol to an 'ideal world' in which there exists a trusted third party (the 'ideal Phish Market functionality') that is completely trusted by both parties. The protocol in the ideal world is much simpler than that in the real world, hence its security guarantees are easier to understand intuitively. We prove our protocol's security by showing that any attacks by the Buyer on the protocol in the real world (without the trusted third party) can be performed in the ideal world as well. Hence, our intuitions for the ideal world must hold for the real world too (this is the ideal/real simulation paradigm).

Below, we describe the ideal-world protocol for a single transaction. In both the real and the ideal world, the Buyer's inputs consist of T, a set of tags in which the Buyer is interested, and U, a set of previously known URLs. The Seller's inputs consist of a tag t and a URL u. The output of the protocol, on the Buyer's side, is the tag t, and optionally the URL u (if the Buyer was interested in it). On the Seller's side, the output is a payment commitment. We denote $\mathrm{Com}_{\mathrm{pk}^*}(x)$ a commitment to a value x.[2]

The protocol in the ideal world proceeds as follows:

1: The ideal functionality waits for the Buyer to send U and the Seller to send t, u.

[2] For clarity, we're ignoring the fact that the commitments are randomized — the commitment function is actually $\mathrm{Com}_{\mathrm{pk}^*}(x, r)$, where r is the commitment's randomizer).

2: The functionality then sends t to the Buyer and waits for the Buyer to respond.
3: **if** The Buyer responds with 0 (she's interested in t) **then**
4: The functionality sends u to the Buyer.
5: **if** $u \notin U$ **then**
6: The functionality sends $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ to the Seller.
7: **else** // $u \in U$
8: The functionality sends $e = \mathrm{Com}_{\mathrm{pk}^*}(0)$ to the Seller.
(The functionality will allow a corrupt Buyer to send $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ in this case as well)
9: **else** // *The Buyer is not interested in t*
10: The functionality sends $e = \mathrm{Com}_{\mathrm{pk}^*}(0)$ to the Seller.

We allow a corrupt party to abort the computation at any point, in which case the other party will receive a special $\perp$ symbol from the ideal functionality (this corresponds to a cheating party being detected). This ideal-world protocol is very similar to the optimal ideal-world protocol described in the beginning of this section. However, in this protocol the ideal party always sends the tag to the Buyer, and if the Buyer is interested, always sends the URL to the Buyer (rather than only sending those URLs that were both interesting and not previously known). This extra 'information leakage' (compared to the optimal protocol) is the result of allowing the Buyer to perform the database lookup on her own.

Formally, the Seller's security is defined as follows:

Theorem 2 (Seller's Security). *For any set of inputs to the Buyer and Seller, and for every (probabilistic polynomial-time) adversary $\mathcal{A}$ that corrupts the Buyer in the real world, there exists a simulator $\mathcal{S}$ who corrupts the Buyer in the ideal world such that the outputs of both parties in the ideal world (the ideal-world Seller and $\mathcal{S}$) are computationally indistinguishable from the outputs of both parties in the real world (the real-world Seller and $\mathcal{A}$), under the assumption that the underlying cryptographic primitives are secure.*

The intuitions behind the proof of this theorem appear in Appendix A.2

Side-Channel Attacks. As with every 'provably secure' system, the proof of security only holds as long as certain assumptions are met. For example, it may be possible to break the security of the protocol if the parties receive information outside the 'legitimate' channels specified by the protocol (these unanticipated information channels are called *side channels*).

The Phish-Market protocol is potentially vulnerable to a timing side-channel attack: the Seller can measure the time it takes the Buyer to complete a transaction. If this time depends on whether or not she was interested in the tag, or on whether or not she already knew the URL, the Seller will gain information about the Buyer's client list or coverage rate. This particular attack can be foiled with relatively little effort by adding artificial delays to the code to ensure all code paths on the Buyer's side take the same time[3]. Of course, as in the case of any secure protocol, the Phish-Market protocol may be vulnerable to other side-channel attacks that we did not anticipate.

[3] Note that the delays are not random noise — the delay on each code path must be computed so that the total time taken by the Buyer does not depend on her input.

2.3 Formal Protocol Definition

We give a full protocol listing (in pseudocode) below. We describe separately the pseudocode for the Sellers' and Buyers' sides of the protocol. To make the protocol listing easier to read, we divide it into a number of smaller subprotocols (called as subroutines from the top-level protocol, Prot. 1). Prot. 1a specifies the top-level protocol run by the Seller and Prot. 1b that run by the Buyer. Prot. 2 is called by the Buyer when she is interested in the tag sent by the Seller (the Seller's side of the protocol looks the same whether or not the Buyer was interested). Prot. 3 is used to prove knowledge of a given commitment value; this protocol has three "sides": Prot. 3a is the Seller's view of the proof (verification), Prot. 3b is the Buyer's view when performing a "real" proof, and Prot. 3c is the Buyer's view when performing a "fake" proof (using the protocol's trapdoor key).

Throughout the protocol, we denote a Pedersen commitment under public-key pk to a value x and with randomizer r by $\mathrm{Com}_{\mathrm{pk}}(x, r)$. To simplify the description, we assume all the Pedersen commitments are over some group with prime order p (thus, the inputs to the function $\mathrm{Com}_{\mathrm{pk}}$ are both elements of $\mathbb{Z}_p$). We also assume the two parties have previously agreed on a Pedersen public-key pk^* that is binding to both parties (i.e., neither party knows its secret key). See App. B for more details on the cryptographic primitives used in the protocol, including a protocol that can be used to generate Pedersen commitment keys with trapdoors (Prot. 5).

Protocol 1a Phish Market Protocol: Seller

Input: Commitment public-key, pk^*, such that Buyer does not know corresponding secret key.
Input: A URL, u, with tag, t

1: Perform Commitment Key Generation (e.g., Prot. 5a).
Denote the resulting commitment keys $(\mathrm{pk}_0, \mathrm{pk}_1)$ and the secret k // *Learning k will allow Buyer to compute both sk_0 and sk_1*
2: Perform Commitment Key Generation (e.g., Prot. 5a).
Denote the resulting commitment keys $(\mathrm{pk}_2, \mathrm{pk}_3)$ (discard the secret).
3: Wait to receive Merkle root c_U from Buyer // *Root of a Merkle hash tree whose leaves are commitments to known URLs*
4: Choose $r \in_R \mathbb{Z}_p$.
Send $(t, \mathrm{Com}_{\mathrm{pk}^*}(H(u), r))$ to Buyer.
5: Perform OT protocol as sender (Buyer as receiver) with input strings $s_0 = (u, r)$ and $s_1 = k$.
6: Wait to receive commitment e from Buyer. // *'payment' commitment*
Verify that $e \in C$.
7: Wait to receive bit b from Buyer. // *Payment proof based on binding of Com_{pk_b}*
8: Verify that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_b}$ for coin-flipping (Prot. 3a). // *Proves that Buyer can either open e to 1 or knows sk_b*
9: Wait to receive bit $b' \in \{2, 3\}$ from Buyer. // *Payment validity proof based on binding of $Com_{pk_{b'}}$*
10: Verify that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_{b'}}$ for coin-flipping (Prot. 3a).
11: Verify that $e = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{5-b'}}$ for coin-flipping (Prot. 3a). // *Together with previous step proves that Buyer can open e to either 0 or 1*
12: Wait to receive commitment c_u from Buyer // *Buyer's 'previously known commitment' to u*
Verify that $c_u \in C$.
13: Let $c_{\mathrm{test}} \leftarrow \frac{\mathrm{Com}_{\mathrm{pk}^*}(H(u), r)}{c_u}$.
Verify that $c_{\mathrm{test}} = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{1-b}}$ for coin-flipping (Prot. 3a) // *Proves that either Buyer can open c_u to $H(u)$ or that Buyer knows sk_{1-b}*
14: Verify proof that c_u is in set committed to by c_U (e.g. verify a Merkle path from c_u to c_U).

Protocol 1b Phish Market Protocol: Buyer

Input: Commitment public-key, pk^*
Input: Set of commitments to known URLs: $U = \left\{c_{u_1} = \mathrm{Com}_{\mathrm{pk}^*}(H(u_1), r_1), \ldots, c_{u_{|U|}} = \mathrm{Com}_{\mathrm{pk}^*}(H(u_{|U|}), r_{|U|})\right\}$
Input: Set of wanted tags, T

Perform Commitment-Generation (Prot. 5b) with input bit b.
Denote the resulting commitment keys pk_0, pk_1 and sk_b.
Perform Commitment-Generation (Prot. 5b) with input bit $b' \in \{2, 3\}$.
Denote the resulting commitment keys pk_2, pk_3 and $\mathrm{sk}_{b'}$.
Generate a 'chaff' commitment: $c_{\mathrm{chaff}} \in_R C$.
Let $U' \leftarrow U \cup \{c_{\mathrm{chaff}}\}$.
Send $\mathrm{Com}_{\mathrm{set}}(U')$ to Seller (e.g. the root of a Merkle hash tree with elements of U' as the leaves) // *Commitment to set of already known URLs*
Wait to receive $(t, c_{u'})$ from Seller // *Tag and commitment to URL*
if $t \in T$ **then** // *Buyer is interested in tag*
    Run Subprotocol 2
else // *Buyer is not interested in tag*
    Perform OT protocol as receiver (Seller as sender) with choice bit 1.
    Denote result $\mathrm{sk}_0, \mathrm{sk}_1$
    Choose $r_e \in_R \mathbb{Z}_p$.
    Send $e = \mathrm{Com}_{\mathrm{pk}^*}(0, r_e)$ to Seller // *'Fake' payment*
    Send b to Seller // *Use Com_{pk_b} for payment proof*
    'Prove' that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_b}$ and sk_b (Prot. 3c).
    Send b' to Seller // *Use $Com_{pk_{b'}}$ for payment validity proof*
    'Prove' that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_{b'}}$ and $\mathrm{sk}_{b'}$ (Prot. 3c).
    Prove that $e = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{5-b'}}$ and r_e (Prot. 3b).
    Choose a 'chaff' commitment $c_u \in U$
    Send c_u to Seller.
    Let $c_{\mathrm{test}} \leftarrow \frac{c_{u'}}{c_u}$.
    'Prove' that $c_{\mathrm{test}} = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{1-b}}$ and sk_{1-b} (Prot. 3c).
Prove that c_u is in set committed to by $\mathrm{Com}_{\mathrm{set}}(U)$ (e.g. show a Merkle path from c_u to c_U).

Protocol 2 Phish Market Subprotocol: Buyer is Interested in t

Perform OT protocol as receiver (Seller as sender) with choice bit 0.
Denote result u, r'
if $c_{u'} = \mathrm{Com}_{\mathrm{pk}^*}(H(u), r')$ and $\exists i : c_{u_i} \in U$ and $u_i = u$ **then** // *Buyer already knows u*
    Choose $r_e \in_R \mathbb{Z}_p$.
    Send $e = \mathrm{Com}_{\mathrm{pk}^*}(0, r_e)$ to Seller // *'Fake' payment*
    Send b to Seller // *Use Com_{pk_b} for payment proof*
    'Prove' that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_b}$ and sk_b (Prot. 3c).
    Send b' to Seller // *Use $Com_{pk_{b'}}$ for payment validity proof*
    'Prove' that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_{b'}}$ and $\mathrm{sk}_{b'}$ (Prot. 3c).
    Prove that $e = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{5-b'}}$ and r_e (Prot. 3b).
    Let $c_u = \mathrm{Com}_{\mathrm{pk}^*}(H(u), r)$ such that $c_u \in U$.
    Send c_u to Seller.
    Let $c_{\mathrm{test}} \leftarrow \frac{c_{u'}}{c_u} = \mathrm{Com}_{\mathrm{pk}^*}(0, r' - r)$.
    Prove that $c_{\mathrm{test}} = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{1-b}}$ and $r' - r$ (Prot. 3b).
else // *Buyer did not know u or Seller is cheating*
    Choose $r_e \in_R \mathbb{Z}_p$.
    Send $e = \mathrm{Com}_{\mathrm{pk}^*}(1, r_e)$ to Seller // *'Real' payment*
    Send $1 - b$ to Seller // *Use $Com_{pk_{1-b}}$ for payment proof*
    Prove that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_{1-b}}$ and r_e (Prot. 3b).
    Send $5 - b'$ to Seller // *Use $Com_{pk_{5-b'}}$ for payment validity proof*
    Prove that $e = \mathrm{Com}_{\mathrm{pk}^*}(1)$ using $\mathrm{Com}_{\mathrm{pk}_{5-b'}}$ and r_e (Prot. 3b).
    'Prove' that $e = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_{b'}}$ and $\mathrm{sk}_{b'}$ (Prot. 3c).
    Send c_{chaff} to Seller.
    Let $c_{\mathrm{test}} \leftarrow \frac{c_{u'}}{c_{\mathrm{chaff}}}$.
    'Prove' that $c_{\mathrm{test}} = \mathrm{Com}_{\mathrm{pk}^*}(0)$ using $\mathrm{Com}_{\mathrm{pk}_b}$ and sk_b (Prot. 3c).

3 Performance evaluation

3.1 Theoretical efficiency

The advantage of this protocol over a generic secure-computation is its efficiency. We measure efficiency in terms of both computation and communication overhead. In Sec-

Protocol 3a Proof of Committed Value: Seller

Input: Commitment c and claimed value x // *Commitment uses public key* pk^*
Input: Trapdoor Commitment public key pk // *Used for coin flipping*
1: Wait to receive c_{chal} from Buyer.
2: Wait to receive (b, c_b) from Buyer.
Verify that $c_b \in C$.
3: Choose $\text{chal}_1 \in_R \mathbb{Z}_p$
Send chal_1 to Buyer.
4: Wait to receive $(\text{chal}_0, r_{\text{chal}})$ from Buyer.
Verify that $c_{\text{chal}} = \text{Com}_{\text{pk}}(\text{chal}_0, r_{\text{chal}})$.
5: Wait to receive r' from Buyer.
Let $\text{chal} \leftarrow \text{chal}_0 + \text{chal}_1$.
Verify that $c^{\text{chal}} \cdot c_b = \text{Com}_{\text{pk}^*}(\text{chal} \cdot x + b, r')$.

Protocol 3b Proof of Committed Value: Buyer

Input: Commitment $c = \text{Com}_{\text{pk}^*}(x, r_x)$, r_x and claimed value x // *Commitment uses public key* pk^*
Input: Trapdoor Commitment public key pk // *Used for coin flipping*
1: Choose $\text{chal}_0 \in_R \mathbb{Z}_p$ and $r_{\text{chal}} \in_R \mathbb{Z}_p$.
Send $\text{Com}_{\text{pk}}(\text{chal}_0, r_{\text{chal}})$ to Seller.
2: Choose $b \in_R \mathbb{Z}_p$ and $r_b \in_R \mathbb{Z}_p$
Send $(b, \text{Com}_{\text{pk}^*}(b, r_b))$ to Seller
3: Wait to receive chal_1 from Seller.
4: Send $(\text{chal}_0, r_{\text{chal}})$ to Seller.
5: Let $\text{chal} \leftarrow \text{chal}_0 + \text{chal}_1$.
Compute r' such that $c^{\text{chal}} \cdot \text{Com}_{\text{pk}^*}(b, r_b) = \text{Com}_{\text{pk}^*}(\text{chal} \cdot x + b, r')$. // r' *can be efficiently computed using* r_b *and* r_x.
Send r' to Seller.

tion 3.2, we describe our implementation of the protocol, which is instantiated using Pedersen commitments and the Naor-Pinkas OT protocol. To get a theoretical estimate of the protocol's efficiency we count the most expensive operations — those that dominate the protocol's overall cost.

Exponentiations are the most expensive computation required, while the main communications requirement is for parties to exchange several group elements and hashes. For each URL transmitted, the Seller must compute 34 exponentiations, while transmitting 10 group elements and 2 hashes to the Buyer. Meanwhile, the Buyer's computation load is a bit lighter but the communications requirements are slightly higher. The Buyer computes just 24 exponentiations, in addition to sending 39 group elements and $\log|U| + 1$ hashes to the Seller. The complete costs, broken down according to each protocol component, are given in Table 2.

3.2 Implementation performance

We implemented an elliptic-curve (EC) based version of the protocol in Java, using the Bouncy Castle Crypto API[4] for basic EC operations. In our implementation the entire Merkle hash-tree was kept entirely in memory (rather than on disk). This is feasible even for moderately large URL lists (e.g., in one of the experiments the tree consisted of about 18 000 URLs).

Our experiments used the NIST-recommended EC curve P-256 [15] as the group over which both the Pedersen commitments and Naor-Pinkas OT were implemented, and SHA-1 in place of a "random oracle". Both sides of the protocol were simulated on a single server with one dual-core 2.4GHz Intel Xeon processor and 2GB of memory (the main bottleneck in the protocol is CPU — one transaction requires less than

[4] `http://www.bouncycastle.org/`

Protocol 3c Fake Proof of Committed Value: Buyer

Input: Commitment c and claimed value x // *Commitment uses public key* pk^*
Input: Trapdoor Commitment public and secret keys pk, sk // *Used to fake coin flipping*
1: Choose $r'_{\text{chal}} \in_R \mathbb{Z}_p$.
Let $c_{\text{chal}} \leftarrow \text{Com}_{\text{pk}}(0, r'_{\text{chal}})$.
Send c_{chal} to Seller. // *Using sk and* r'_{chal}, *Buyer can open* c_{chal} *to any value*
2: Choose chal $\in_R \mathbb{Z}_p$, $b \in_R \mathbb{Z}_p$ and $r' \in_R \mathbb{Z}_p$.
Let $c_{\text{target}} \leftarrow \text{Com}_{\text{pk}^*}(\text{chal} \cdot x + b, r')$.
Let $c_b \leftarrow \frac{c_{\text{target}}}{c_{\text{chal}}}$.
Send (b, c_b) to Seller. // *Buyer does not know how to open* c_b
3: Wait to receive chal_1 from Seller.
4: Let $\text{chal}_0 \leftarrow \text{chal} - \text{chal}_1$.
Compute r_{chal} s.t. $c_{\text{chal}} = \text{Com}_{\text{pk}}(\text{chal}_0, r_{\text{chal}})$. // r_{chal} *can be efficiently computed using sk and* r'_{chal}
Send $(\text{chal}_0, r_{\text{chal}})$ to Seller.
5: Send r' to Seller. // c_b *and b were computed at step 2 such that* $c^{chal} \cdot c_b = Com_{pk^*}(chal \cdot x + b, r')$

protocol	comp. cost exponentiations	communication cost group elements	communication cost hashes
Seller			
5a	2	2	0
OT	4	2	2
3a	24	4	0
1a	4	2	0
Seller total	34	10	2
Buyer			
5b	2	2	0
OT	2	1	0
3b/3c	16	24	0
1a	4	2	$\log \lvert U \rvert + 1$
Buyer total	24	39	$\log \lvert U \rvert + 1$

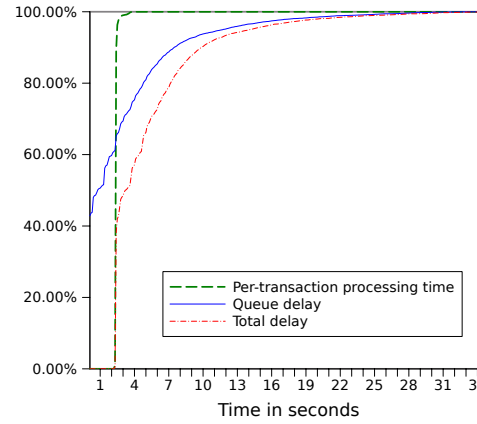

Fig. 2. Theoretical computation and communication costs of the Phish Market protocol (left); observed cumulative distribution function of the time required to share each phishing URL (right).

3kB 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 URL feeds from two large take-down companies during the first two weeks of April 2009. We assigned one of the take-down companies to be the Seller, while making the other the Buyer (we ran experiments using both assignments). For the two-week sample period, one company found 8 582 unique URLs while the other discovered 17 721 URLs. 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 the authors.

The primary metric we use to measure the performance of our implementation is the time required to process and transmit each phishing URL from the seller to the buyer. The less time required for processing URLs, 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 sec-

onds. 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 other URLs to be processed, leading to a longer delay. While a multi-threaded 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 the protocol implementation was optimized for clarity and generality of the source code rather than speed. The average queue delay caused by waiting on other URLs to finish processing was 2.76 seconds, while the longest delay was 34.6 seconds.

To get a better feel for how the processing time varies, Figure 2 (right) 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. 48.4% of URLs were processed in under 3 seconds, yet 9.7% 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 to be removed!

In addition to the total delay (red dash-dot line), Figure 2 (right) plots the two key components of delay. The green dash 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 stretched tail of the overall delay. Hence, if the queue delay were reduced by using multiple processors or threads, the total delay might approach the consistently shorter processing time.

4 Related Work

Sharing Attack Data The academic work on phishing has been diverse, with a useful starting point being Jakobsson and Myers' book [10]. However, there has been only limited examination of the take-down process employed by the banks and specialist companies, even though it is the primary defense employed today. Moore and Clayton estimated the number and lifetimes of phishing websites and demonstrated that timely removal reduced user exposure [12]. Subsequently, they presented evidence (repeated in Section 1) showing that take-down companies do not share data on phishing websites with each other, and they calculated that website lifetimes might be halved if companies shared their URL feeds. They appealed to the greater good in advocating that take-down companies voluntarily exchange URL feeds with each other at no charge. By contrast, this paper proposes a mechanism for sharing where net contributors are compensated by net receivers of phishing URLs.

Information sharing has long been recognized as necessary for improving information security. Gordon and Ford discussed early forms of sharing in the anti-virus industry and contrasted it with sharing when disclosing vulnerabilities [9]. Some have worried that firms might free-ride off the security expenditures of other firms by only 'consuming' shared security information (e.g., phishing feeds) and never providing any data of their own [8], while others have argued that there can also be positive economic incentives for sharing security information [6].

Cryptographic Protocols The Phish Market protocol is an instance of *secure multiparty computation* (MPC). MPC has been a major area of work in theoretical cryptography, and general techniques are known for securely computing any functionality [17,7,3,1].

These techniques, however, are not practical for computing functions that have large input size (e.g., an optimized implementation of Yao's protocol for general two-party computation can take seconds to evaluate a simple function with 32-bit inputs [11]). In our case, one of the inputs to the function is a database of previously-known URLs containing thousands of entries, making general techniques completely impractical.

For many specific functionalities, efficient protocols are known. We use some of these as subroutines in our protocol. We make use of the Naor-Pinkas OT protocol [14], which is itself a more efficient version of the Bellare-Micali OT protocol [2]. We also use a generalization of the Chaum-Pedersen protocol for proving in zero-knowledge the value of a commitment [5].

5 Concluding remarks

Sharing data between competing firms is hard. Yet security mechanisms are becoming increasingly data-driven, from identifying malware hosts to blocking spam and shutting down phishing websites. Consequently, sharing data is now essential as no single defender has a complete view of attacker behavior.

In this paper, we have devised a mechanism to make it easier for take-down companies to interact: by compensating net contributors of phishing URLs, we can bolster the incentive to share while rewarding investment into better discovery techniques. As a bonus, our protocol has the desirable property of being provably secure and allowing parties to share data without relying on a trusted third party to mediate. Crucially, the protocol is also efficient: our elliptic-curve-based implementation easily processed the phishing URLs in a two-week sample from two take-down companies while introducing average delays of 5 seconds before sharing.

Of course, to cut phishing website lifetimes in half and reduce the annual financial exposure due to phishing by several hundred million dollars, we must still convince the take-down companies that sharing is a good idea. We feel that the security guarantees our protocol provides will make it easier for companies to at least explore the idea of sharing data with their competitors. At present, many companies still cling to the view that their feed is best. Fortunately, our protocol offers companies the chance to put their claims to the test while avoiding the potential for public embarrassment if they happen to find that sharing can indeed help.

References

1. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC '90*, pages 503–513, New York, NY, USA, 1990. ACM.
2. M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 547–557, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

3. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *STOC '88*, pages 1–10, pub-ACM:adr, 1988. ACM Press.
4. M. Blum. Coin flipping by telephone - A protocol for solving impossible problems. In *Proceedings of the 25th IEEE Computer Society International Conference*, pages 133–137, 1982.
5. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO '92*, pages 89–105, London, UK, 1993. Springer-Verlag.
6. E. Gal-Or and A. Ghose. The economic incentives for sharing security information. *Information Systems Research*, 16(2):186–208, 2005.
7. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — A completeness theorem for protocols with honest majority. In ACM, editor, *STOC '87*, pages 218–229, pub-ACM:adr, 1987. ACM Press.
8. L. Gordon, M. Loeb, and W. Lucyshyn. Sharing information on computer systems security: An economic analysis. *Journal of Accounting and Public Policy*, 22(6):461–485, 2003.
9. S. Gordon and R. Ford. When worlds collide: information sharing for the security and anti-virus communities, 1999. IBM research paper.
10. M. Jakobsson and S. Myers, editors. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley, New York, 2006.
11. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
12. T. Moore and R. Clayton. Examining the impact of website take-down on phishing. In *Anti-Phishing Working Group eCrime Researchers Summit (APWG eCrime)*, pages 1–13, 2007.
13. T. Moore and R. Clayton. The consequence of non-cooperation in the fight against phishing. In *Anti-Phishing Working Group eCrime Researchers Summit (APWG eCrime)*, pages 1–14, 2008.
14. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA '01*, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
15. NIST. Digital signature standard (DSS). FIPS 186-2, January 2000. Available at URL: `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`.
16. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, volume 576, pages 129–140, 1991.
17. A. C.-C. Yao. How to generate and exchange secrets. In *FOCS '96*, pages 162–167, Los Alamitos, CA, USA, 1986. IEEE Computer Society.

A Security Proof Sketches

A.1 Proof Sketch for Theorem 1

The Seller's view of the protocol consists of the its input, its random coins and the messages received from the Buyer (the view explicitly excludes the total payment received at the end of a period). Going over the messages received from the Buyer, we verify that they are statistically independent of the Buyer's inputs:

1. The public keys generated by the commitment key-generation protocol are statistically indistinguishable. In particular, the Buyer's input to these protocols is independent of the Seller's view.

2. The commitments c_U, e and c_u are statistically hiding, hence do not depend on the messages committed or on each other. The randomness used in the commitments is independent of the Buyer's inputs.
3. The bits b' and b are independently and uniformly distributed (note that the XOR of these bits with the Buyer's input to the commitment key-generation protocols do depend on the Buyer's input, however no additional information about these bits is sent).
4. The OT protocol (executed in step 5 or Protocol 1a) is statistically hiding.
5. Protocol 3 is a statistical zero-knowledge proof of knowledge (assuming we use the Fiat-Shamir heuristic to make it non-interactive), hence the view of the Seller is statistically independent of the Buyer's inputs to this protocol.
6. The proof of membership in step 14 consists of a random set of hashes (the path in the Merkle tree) which are statistically hiding (assuming we use a random oracle). The length of the path does reveal information about the size of the set U (but that is allowed by the security definition).

Note that this proof assumes the protocol is only run once. In practice, it will be run multiple times without recomputing all the leaves of the Merkle hash tree. Hence, for security to hold the Seller should not be allowed to resend a URL it previously sent (in this case, c_u will be a new, independent value in each of the Seller's views). Even if this is done, however, reusing the Merkle tree can cause a small loss of privacy for the Buyer — in particular, the seller may gain some information about the number of leaves added to the tree between invocations of the protocol. We leave the detailed analysis of this case to the full version of the paper.

A.2 Proof Sketch for Theorem 2

The proof of the theorem is by constructing the simulator $\mathcal{S}$ and showing that its output is indistinguishable from that of the real-world adversary $\mathcal{A}$. We follow the standard scheme for proofs in the ideal/real paradigm: $\mathcal{S}$ simulates a real-world execution using $\mathcal{A}$ as a black box, simulating the random oracle H and the honest Seller, and outputting anything $\mathcal{A}$ outputs.

$\mathcal{S}$ simulates the Seller by simulating Protocol 1a exactly as an honest Seller would, with the following modifications:

1. If $\mathcal{A}$ aborts at any time, or if any of the Seller's verifications fail, $\mathcal{S}$ sends the abort command to the ideal functionality.
2. $\mathcal{S}$ must send a set U to the ideal functionality before it can receive the tag t (which it needs to simulate step 4 in Protocol 1a). To extract U, $\mathcal{S}$ makes use of the random oracle. Since $\mathcal{S}$ is simulating the oracle for $\mathcal{A}$, it can record any queries made to the oracle. $\mathcal{S}$ runs the simulation until $\mathcal{A}$ sends the commitment c_U (in step 3 of Protocol 1a). $\mathcal{S}$ then lets U be the set of all queries $\mathcal{A}$ made to H up to that point. Since the commitment c_U is to the a set of the form $\mathrm{Com}_{\mathrm{pk}^*}(H(x), r)$, if $\mathcal{A}$ did not query H about a particular string before sending c_U, it will not be able to open the commitment correctly later. Note that the set of queries to H may be larger than the set committed to by c_U; however, this will not hurt the simulation since $\mathcal{S}$ can always choose to send a 'real' payment ($e = \mathrm{Com}_{\mathrm{pk}^*}(1, r)$) even when $u \in U$.

3. In step 4, $\mathcal{S}$ chooses a random value h for $H(u)$ (it doesn't know u yet). Since $\mathcal{S}$ is simulating the random oracle, it will later be able to claim the preimage of h is u, for any u.
4. To run the OT protocol in step 5, $\mathcal{S}$ must decide on an input u to the OT protocol. Here we use the fact that the OT protocol also defines security for the Sender in the ideal/real simulation paradigm. Thus, there exists a simulator $\mathcal{S}_{\text{OT}}$ that runs in a world with an ideal OT functionality and can simulate a corrupt Chooser. $\mathcal{S}$ runs $\mathcal{S}_{\text{OT}}$ and simulates the ideal OT functionality with $\mathcal{A}$ as the Chooser. $\mathcal{S}_{\text{OT}}$ must extract the Chooser's choice bit and send it to the ideal OT functionality in order to correctly simulate the Chooser, hence $\mathcal{S}$ will learn this bit. $\mathcal{S}$ sends this choice bit as the Buyer's response to the ideal Phish Market functionality. If the choice bit was 0, $\mathcal{S}$ receives u from the ideal functionality and can run the OT protocol with the correct u. If the choice bit was 1, it uses a random value for u, but the security of the OT protocol ensures that the Buyer will not be able to distinguish between a random value and the correct u.
5. In order to learn the randomness r used in the commitment e, $\mathcal{S}$ makes use of the fact that Protocol 3 is a proof of knowledge (when the commitment used for coin flipping is binding). Thus, there exists a knowledge extractor that, given access to a Buyer who can prove that a commitment e can be opened to a value x, can output the opening of the commitment e. Note that of the two proofs in steps 10 and 11, at least one must use a commitment that is binding for the Buyer (due to the properties of the commitment key generation protocol). Hence, the knowledge extractor will be able to run in at least one of the instances, allowing $\mathcal{S}$ to learn r (and whether e is a commitment to 0 or to 1).
6. If $\mathcal{S}$ learned u and $u \in U$ but e is a commitment to 1, $\mathcal{S}$ instructs the ideal functionality to send the Seller a commitment to 1 rather than to 0.

To complete the proof, we must show that the output of $\mathcal{S}$, together with the output of the ideal Seller, is indistinguishable from the output of $\mathcal{A}$ together with the output of the real-world Seller (in a real execution of the protocol). The main problem that remains is showing that when $\mathcal{S}$ chose to learn u from the ideal functionality and $u \notin U$, then the commitment e received from the Buyer is a commitment to 1 (otherwise the output of the ideal-world Seller will not match the output of the simulated Seller). This must be the case because at least one of the proofs in steps 8 and 13 must use a commitment that is binding to the Buyer (since it chose to receive u rather than k in the OT protocol, hence either Com_{pk_b} or $\text{Com}_{\text{pk}_{1-b}}$ is binding). If the proof in step 8 uses the binding commitment, the existence of the knowledge extractor guarantees $\mathcal{S}$ can open e to a value of 1. If the proof in step 13 uses the binding commitment, $\mathcal{S}$ can use the knowledge extractor to extract an opening of c_u to $H(u)$. Since $u \notin U$, $H(u)$ was not the response to a query sent to H before the Buyer computed c_U. However, the probability that the Buyer guessed $H(u)$ without querying the oracle is negligible. Hence, either we can use the buyer to open c_u in two different ways, or we can use the Buyer to break the set commitment scheme ($\mathcal{S}$ can rewind the Buyer and set $H(u)$ to a different value – then the buyer must either give a new different value for c_u or a different opening of c_u).

B Cryptographic Building Blocks

The Phish Market protocol requires several cryptographic primitives with specific properties. In this section we describe them in more detail:

Commitments and Trapdoor Commitments. A commitment scheme is a two-party functionality; one party is called the *committer* and the other is the *receiver*. A commitment under public key pk is a function $\mathrm{Com}_{\mathrm{pk}} : \mathcal{M} \times \mathcal{R} \mapsto \mathcal{C}$, mapping a message $m \in \mathcal{M}$ and randomizer $r \in \mathcal{R}$ to $\mathcal{C}$ (to simplify the presentation, below we will use $\mathcal{M} = \mathcal{R} = \mathbb{Z}_p$, the group of integers mod p for some prime p; this is not a requirement of the protocol, however). The committer *opens* the commitment by revealing both the message and the randomizer used to create it. Commitments must satisfy two security properties:

- *Hiding.* For any public key pk and any two messages x, y, $\mathrm{Com}_{\mathrm{pk}}(x, r)$ is indistinguishable from $\mathrm{Com}_{\mathrm{pk}}(y, r)$ when r is chosen uniformly at random. In our protocol, we use commitments that are *statistically* hiding: the indistinguishability holds even for a computationally unbounded adversary.
- *Binding.* For any computationally bounded adversary $\mathcal{A}$ without knowledge of the secret key, the probability that $\mathcal{A}$ outputs (x, r) and (y, r') such that $\mathrm{Com}_{\mathrm{pk}}(x, r) = \mathrm{Com}_{\mathrm{pk}}(y, r')$ is negligible (more formally, such an adversary can be used to efficiently extract the secret key corresponding to pk).

For our protocol, we require a special type of commitment scheme, a *Trapdoor Homomorphic Commitment* that has two additional properties:

- *Homomorphic.* The message space $\mathcal{M}$ must be a group. Given a public key pk and any two commitments $c_1 = \mathrm{Com}_{\mathrm{pk}}(x, r_1), c_2 = \mathrm{Com}_{\mathrm{pk}}(y, r_2)$, it is possible to compute $c_3 = \mathrm{Com}_{\mathrm{pk}}(x + y, r_3)$. Moreover, a party that knows x, r_1, y and r_2 can compute r_3. To simplify, we will assume the range of the commitment, $\mathcal{C}$, is a group, and that $\mathrm{Com}_{\mathrm{pk}}(x, r_1) \cdot \mathrm{Com}_{\mathrm{pk}}(y, r_2) = \mathrm{Com}_{\mathrm{pk}}(x + y, r_1 + r_2)$.
- *Trapdoor.* For any public key pk, any commitment $c = \mathrm{Com}_{\mathrm{pk}}(x, r)$ and any message y, it is possible to efficiently compute r' such that $c = \mathrm{Com}_{\mathrm{pk}}(y, r')$ using sk, x and r. Moreover, when r is chosen uniformly at random, r' is statistically close to uniform.

We also require the commitment scheme to have a special key-generation protocol that generates a pair of public-keys $(\mathrm{pk}_0, \mathrm{pk}_1)$, such that the committer provably learns only one of the corresponding secret keys, but the receiver gains no information about which of the keys is known to the committer. The receiver should learn an 'enabling key' that, when given to the committer, allows the committer to recover both secret keys.

In this paper, we assume the commitments used are Pedersen Commitments [16]. A full description (including a protocol for key-generation) appears in App. B.1.

1-out-of-2 String Oblivious Transfer (OT). OT is a two-party functionality, with a *sender* and a *receiver*. The sender's inputs consist of two strings s_0 and s_1, while the receiver's input is a single bit, c. An OT protocol ensures that the receiver gets s_c but no information about s_{1-c}, while the sender gets no information about c. In this paper we assume we are using the Naor-Pinkas OT scheme.

Random Oracle. The random oracle is a hash function H that is modeled in the security analysis as a random function. In practice, H would be an explicit cryptographic hash function such as SHA-1. Although we can replace some uses of the Random Oracle with weaker cryptographic functionalities (such as collision-resistant hash functions), to simplify the description and analysis of the protocol we do not make this separation here.

Set Commitment. This is a commitment scheme that allows a committer to commit to a set of strings (rather than a single message), and later prove that a string belongs to the set without revealing anything about the other strings in the set. In this paper we assume we are using a Merkle hash tree.

B.1 Pedersen Commitments

Pedersen commitments are based on the hardness of discrete log in a group $\mathcal{G}$ of prime order p. The commitment public key is a pair of generators $g, h \in \mathcal{G}$, and the corresponding secret key is $\log_g h$. We will assume g is fixed in advance and common to all the public keys. To commit to a message $m \in \mathbb{Z}_{|\mathcal{G}|}$, the committer chooses $r \in_{\mathcal{R}} \mathbb{Z}_p$ and sends $\mathrm{Com}_{g,h}(m, r) = g^m h^r$. This commitment is perfectly hiding (for any message m, $\mathrm{Com}_{g,h}(m, r)$ is uniformly distributed in $\mathcal{G}$ when r is chosen randomly). It is also computationally binding: given $m \neq m'$ and r, r' such that $c = g^m h^r = g^{m'} h^{r'}$, it is easy to compute $\log_g h \equiv \frac{m-m'}{r'-r} \pmod{p}$.

Computing (or verifying) a Pedersen commitment requires two exponentiations in the group $\mathcal{G}$. Committing and opening a commitment (to a previously known value) require the committer to send a single group element.

Binding Key Generation. Since the order of the group $\mathcal{G}$ is prime, every element of $\mathcal{G}$ (except the identity) is a generator of the group. To choose a public-key that is binding for both parties, the parties can use a variation of Blum's 'coin-flipping over the telephone' [4] to choose a random group element. This is described in Protocol 4 (note that we use the random oracle H as a standard commitment in this protocol).

Protocol 4a Pedersen Binding Commitment Key Generation: Seller

Input: (Common) Group $\mathcal{G}$ of prime order p in which DL is hard
Input: (Common) A generator g of $\mathcal{G}$
1: Choose a random $k \in_{\mathcal{R}} \mathbb{Z}_p$.
2: Send $h \leftarrow H(g^k)$ to Buyer.
3: Wait to receive d from Buyer.
4: Send $e \leftarrow g^k$ to the Buyer.
5: Output pk $\leftarrow d \cdot e$ // *If Buyer knows* $\log_g pk$ *she must also know* k

Special Key Generation. Protocol 5 gives the required special key-generation protocol for Pedersen commitments (this protocol is based on the idea from the Bellare-Micali [2] OT scheme). An execution of the key generation protocol requires each party to perform a single exponentiation in the group $\mathcal{G}$ and send a single group element. Note that these can be performed ahead of time (since they do not rely on information from the other party).

Protocol 4b Pedersen Binding Commitment Key Generation: Buyer

Input: (Common) Group $\mathcal{G}$ of prime order p in which DL is hard
Input: (Common) A generator g of $\mathcal{G}$
1: Choose a random $k' \in_R \mathbb{Z}_p$.
2: Wait to receive h from Seller.
3: Send $d \leftarrow g^{k'}$ to the Seller
4: Wait to receive e from Seller.
Verify that $h = H(e)$.
5: Output $\mathrm{pk} \leftarrow d \cdot e$ // *If Seller knows* $\log_g pk$ *he must also know* k'

Protocol 5a Pedersen Commitment Special Key Generation: Seller

Input: (Common) Group $\mathcal{G}$ of prime order p in which DL is hard
Input: (Common) A generator g of $\mathcal{G}$
1: Choose a random $k \in_R \mathbb{Z}_p$.
2: Send g^k to Buyer.
3: Wait to receive pk_0 from Buyer.
4: Let $\mathrm{pk}_1 \leftarrow \frac{g^k}{\mathrm{pk}_0}$ // *Ensures Buyer knows either* sk_0 *or* sk_1 *but not both.*
5: Output $(\mathrm{pk}_0, \mathrm{pk}_1)$ and k.

Protocol 5b Pedersen Commitment Special Key Generation: Buyer

Input: (Common) Group $\mathcal{G}$ of prime order p in which DL is hard
Input: (Common) A generator g of $\mathcal{G}$
1: Wait to receive $h \in \mathcal{G}$ from Seller.
2: Choose random $b \in_R \{0, 1\}$
3: Choose random $\mathrm{sk}_b \in_R \mathbb{Z}_p$
4: Compute $\mathrm{pk}_b \leftarrow g^{\mathrm{sk}_b}$ and $\mathrm{pk}_{1-b} \leftarrow \frac{h}{\mathrm{pk}_b}$.
5: Send pk_0 to Seller.
6: Output $(\mathrm{pk}_0, \mathrm{pk}_1)$ and (b, sk_b)