RevCast: Fast, Private Certificate Revocation over FM Radio

Aaron Schulman
Stanford University
aschulm@stanford.edu

Dave Levin
University of Maryland
dml@cs.umd.edu

Neil Spring
University of Maryland
nspring@cs.umd.edu

ABSTRACT

The ability to revoke certificates is a fundamental feature of a public key infrastructure. However, certificate revocation systems are generally regarded as ineffective and potentially insecure: Some browsers bundle revocation updates with more general software updates, and may go hours, days, or indefinitely between updates; moreover, some operating systems make it difficult for users to demand recent revocation data. This paper argues that this sad state of affairs is an inexorable consequence of relying on unicast communication to distribute revocation information.

We present RevCast, a broadcast system that disseminates revocation data in a timely and private manner. RevCast is not emulated broadcast over traditional Internet links, but rather a separate metropolitan-area wireless broadcast link; specifically, we have designed RevCast to operate over existing FM radio, although the principles apply to alternative implementations. We present the design, implementation, and initial deployment of RevCast on a 3 kW commercial radio station using the FM RDS protocol. With the use of two types of receivers (an RDS-to-LAN bridge that we have prototyped and an RDS-enabled smartphone), we show that, even at a low bitrate, RevCast is able to deliver complete and timely revocation information, anonymously, even for receivers who do not receive all packets all the time.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; E.3 [**Data Encryption**]: Public key cryptosystems, Standards

Keywords

Certificates; Revocation; FM Radio; RDS; Heartbleed; Broadcast; Security; X.509

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2660267.2660376.

1. INTRODUCTION

Today's public key infrastructures (PKIs) provide the means by which users can verify with whom they are communicating. A necessary primitive in any PKI is the ability for a certificate authority (CA) to verifiably *revoke* a certificate it has issued, and to *disseminate* these revocations to all users who may have cached the corresponding certificates. In practice, the most common means of certificate revocation is for a CA to aggregate its revocations into *certificate revocation lists* (CRLs), as described in the X.509 standard [6]. CAs are actively issuing revocations in their CRLs. We observed at least one revocation in 17.6% of the 20 second intervals in March 2014 and April 2014.

CRLs have been the focus of debate for nearly a decade: given the high bandwidth costs of disseminating entire lists of revocations to virtually all users, what properties can we hope to achieve [18, 13], and if not, then should we eliminate them altogether [25, 16]? We distill from these public debates on CRLs a set of properties that we believe any revocation system should have:

- **Timeliness**: The time from when a certificate is revoked to the time that all interested, active parties learn of the revocation should be minimized. Ideally, this would be on the order of *seconds*, as opposed to the hours or days it takes clients today.
- **Low-cost dissemination**: The raw bandwidth consumed must scale well with the number of clients, CAs, and certificates, and overall deployment costs must be mitigated.
- **Privacy**: Ideally, obtaining revocation information should not result in a loss of client privacy, e.g., by revealing browsing habits.

These three naturally desirable properties have proven difficult to achieve in tandem. The de facto means of disseminating revocations, CRLs, achieve low cost and privacy, but at the expense of timeliness. Google, for instance, pushes revocation information to Chrome browsers via Chrome software updates [15], though typically infrequently. On the other end of the spectrum, the Online Certificate Status Protocol (OCSP) [26] provides a more timely alternative to CRLs—clients can request revocation state precisely when they visit a given site—but results in users divulging their browsing habits to a typically small set of third parties. Various approaches have been proposed that employ novel techniques in cryptography [13, 10, 19, 17, 8, 1] and distributed systems [31], but have broadly been able to achieve two of the three properties simultaneously. (We review related work more thoroughly Section 6.)

Based on these options, it would appear that revocation systems are faced with inherent trade-offs between timeliness, low cost, and privacy. Indeed, the communication primitives of the Internet are not tailored to achieving what would be ideal for certificate revocation: delivering a constant stream of data to all users all the time. As a result, all revocation systems of which we are aware assume that clients can pull data only infrequently, on the order of hours or days.

In this paper, we propose augmenting Internet communication with a very old communication primitive: metropolitan-area broadcast. We present RevCast, a system which uses low bit-rate radio broadcast, such as FM radio, to deliver revocation information in a manner that is timely, naturally scales to a virtually unbounded number of receivers within a given area, and is inherently receiver-anonymous.

RevCast challenges the established belief that the trade-offs between timeliness, low cost, and privacy must be inherent to all revocation systems. Despite operating over FM's measly *421.8 bit/s*, RevCast users need not wait hours or days for up-to-date revocation data: we show, for the vast majority of the time, users' local revocation state remains consistent within at most tens of seconds. The main technical challenge that RevCast addresses is the ability to compactly represent many CAs' attestations over such a low-bandwidth channel. We demonstrate empirically that RevCast is able to use this meager bit-rate from a *single tower* to simultaneously deliver in a timely manner *all* revocations reported by 621 CAs over the span of a month.

This rest of this paper is organized as follows: We describe in §2 the benefits that broadcast has over unicast when disseminating revocation information, and argue that FM RDS is particularly well-suited for this task. In §3, we present the RevCast design, which makes use of multi-signatures to achieve a compact representation of revocation state while scaling to a large number of CAs. To demonstrate that RevCast permits a practical deployment today, we describe in §4 an inexpensive hardware prototypical receiver, and evaluate it with an initial deployment on a 3 kW commercial radio station. With a trace-based evaluation, we demonstrate in §5 that RevCast's compact representation of revocation state allows it to disseminate up-to-date information from hundreds of CAs, typically within tens of seconds, despite FM RDS's meager bitrate. We also stress-test RevCast with recent spikes in revocation rates due to the Heartbleed OpenSSL vulnerability. Finally, we review related work in §6 and conclude in §7.

Ultimately, RevCast demonstrates that, through the novel use of existing infrastructure, a more timely and more private revocation system is a practical reality.

2. DISSEMINATING REVOCATIONS

The key feature of certificate revocation is that one must be able to retrieve an authoritative, current statement that a certificate has *not* been revoked. Thus, when a certificate is revoked, all users who have received the certificate—or will ever receive it before it expires—must obtain a non-repudiable attestation that it is no longer valid. Because this set of users can be *huge*—especially for widely popular services—the primary challenge in practice is to disseminate the proofs of revocation to all users who need them. In this section, we describe why unicast protocols are fundamentally limited in achieving this goal, why broadcast is necessary, and why we chose to focus on FM RDS as a practical substrate for carrying RevCast.

2.1 Unicast revocation is flawed

The most common forms of certificate revocation are designed around unicast. Commonly, this means either retrieving a complete list (a CRL) to ensure that a certificate of interest is not present, or querying for the status of a specific certificate via OCSP. In both cases, for a current statement, a query must be sent, per-server and disclosing the server of interest in the case of OCSP, and per-CA and requiring storage of unnecessary entries in the list in the case of CRLs.

Security is paramount in the design of a PKI, but implementations of OCSP and CRL checking typically treat a failure to communicate to the server as a benign indication of Internet connectivity issues [3, 14]. This is despite the fact that an attacker capable of intercepting a client's traffic to impersonate a server would almost certainly also be able to intercept and discard OCSP.

Even multicast distribution of revocation information faces similar problems. Application layer multicast is vulnerable to eclipse attacks [27], where a node's peers are replaced by puppets of or Sybils [7] created by the attacker so that the attacker can deny service to or otherwise fool the victim. IP multicast can similarly be intercepted by a man-in-the-middle and may be additionally vulnerable to a denial of service attack by a well-provisioned attacker.

In short, although unicast has been successful as the de facto communication primitive on the Internet, revocation imposes unique requirements—in particular, the need to transmit the same information to virtually all users—for which unicast is ill-suited.

2.2 Broadcast is very well suited to revocation

The broadcast primitive, which delivers the same data to all destinations, is the natural fit for revocations, which must be delivered to (virtually) all clients. Unfortunately, there is no such wide-area broadcast primitive on the Internet today (broadcasts are relegated to subnets), nor is there likely to be in the near future; interdomain broadcast would be difficult to price, and the inherent amplification factor would be a prime target for attackers.

However, *metropolitan radio broadcast*, as an alternate channel to the standard links on the Internet, is in widespread use today. Moreover, it has many properties that make it a vast improvement over unicast: It is robust to the types of eclipse and man-in-the-middle attacks that plague application-layer multicast. While subject to other forms of attack—in particular, jamming—metropolitan-area radio broadcast is generally regulated and monitored by an entity like the FCC, who triangulates sources of jamming and prosecutes, if necessary. Additionally, metropolitan-area broadcast is also trivially receiver-anonymous, and has communication cost that scales with the number of revocations rather than the number of secure connections.

An inherent property of wireless broadcasts is that an increase in transmission range results in lower bit-rates. As a result, metropolitan-area radio broadcast systems typically have *far* less bandwidth than standard Internet links. For example, cell broadcast operates at 5225.7 bit/s across a cell coverage area, and FM RDS (described more below) operates at only 421.8 bit/s yet covers approximately 100 miles. While these bit-rates would be unreasonable for a

single user, the key property that revocation allows us to exploit is that *all* users must obtain the same data. Effectively, then, metropolitan area broadcast achieves a throughput on the order of their (low) bit-rate multiplied by the (huge) number of users within broadcast range. In the case of FM RDS, some towers cover over 10 million people, resulting in an effective bandwidth of 4.22 Gbit/s.

2.3 FM RDS is particularly well suited

The European Broadcasting Union developed the FM Radio Data System (RDS) in the 1980s [23]. The RDS Open Data Application (ODA) framework permits transmission of arbitrary data, limited to an effective *421.8 bit/s*. We chose to focus our design and evaluation on FM RDS for the following reasons.

(1) FM transmissions have extensive coverage. Their frequencies propagate through walls, and their transmission ranges can be approximately 100 miles, depending on transmitter power and terrain. This permits remarkably complete coverage with few transmitters. Conversely, higher frequencies associated with cellular infrastructure require more transmitters to achieve similar coverage, and may be less dependable indoors. Also, the higher frequency and lower signal-to-noise ratio of satellite transmissions makes them unreliable indoors.

(2) FM RDS receivers are inexpensive and small, capable of being embedded into existing devices. They are even present (but often disabled) in today's cell phone chipsets, notably the BCM4334 in the iPhone 5S. Although lower frequency shortwave transmissions offer even better coverage than FM's VHF transmissions, receiving shortwave requires large antennas.

(3) FM RDS transmitters are already licensed to transmit data, and have the necessary Internet connectivity and computing infrastructure to do so. Station identification and information about the currently playing artist and song are currently sent from distribution servers to radio stations via TCP/IP, then broadcast to radios via FM RDS. An alternate design, perhaps using an entire channel or an altogether different subcarrier, would require licensing.

(4) Even considering its low bit-rate, FM RDS is currently under-utilized and under-monetized. It takes relatively few bits to provide artist, title, and station information. Also, based on personal communication with radio station operators, we have found that FM RDS has yet to be monetized; we therefore anticipate that shifting the infrastructure costs that CAs currently face towards FM stations would be a welcomed deployment by station operators.

Moreover, delivering revocation data via FM RDS alongside standard Internet-based dissemination systems provides defense in depth that makes attacks considerably more difficult. To block a user from receiving up-to-date revocation state, an attacker would have to simultaneously intercept Internet traffic *and* create a fault in the victim's FM RDS reception. Likewise, although typical Internet connectivity interruption may cause a CRL fetch or OCSP query to fail, it should not fail simultaneously with an orthogonal, radio-based system.

2.4 Summary

It is our contention that broadcast, particularly on a channel that is independent from one an attacker might control, provides a fundamental, missing primitive that can secure the public key infrastructure efficiently. To move towards this vision, we identify the primary technical challenge to be making efficient use of metropolitan radio broadcast's inherently low bit-rate so that all users can receive revocation information in a timely manner. In the next section, we describe a protocol that shows this to be possible through the novel use of recent cryptographic mechanisms.

3. REVCAST TRANSMISSION PROTOCOL

Our goal is to deliver revocations to millions of end hosts in a manner that is:

- **consistent**: all receivers should be able to determine how out of date their "revocation state" is
- **timely**: so long as CAs follow the protocol, end hosts' revocation state should be up to date within at most tens of seconds
- **privacy-preserving**: our protocol should reveal nothing about users' browsing habits to any party
- **scalable with respect to the number of CAs**: our protocol should permit hundreds of CAs without sacrificing any other properties
- **able to support practical rates of revocation**: our metric for this is the ability to support all revocations from an existing months-long data set of revocations
- **consistent with the existing trust model**: our protocol should remain correct without requiring either users or CAs to trust any additional parties than in the current web PKI

Additionally, we seek to optimize to support users who may occasionally miss small portions of the broadcast. We offer no additional optimizations for handling loss of very large portions of the broadcast; in such events, the user would have to obtain the data he missed via traditional means (e.g., downloading the missed CRLs), but can benefit from broadcasts in the interim.

These properties are natural to desire; indeed, we are not the first to propose them [25, 16, 31]. However, we are the first to demonstrate they can be practically realized using existing infrastructure at low cost.

The principals involved in our protocol consist of CAs who are the source of all revocation state, users who wish to maintain up-to-date revocation state, and *towers* who broadcast this state to users. As we will see, in our design, towers also play a slightly more active role in preparing the revocation state for broadcast, but this does not require any additional trust to maintain correctness.

Conceptually, the RevCast protocol is rather simple: the CAs sign attestations of their revocation state and forward them to the tower, who then broadcasts the attestations. The challenge is to arrive at a compact representation of the revocation state so as to keep transmissions timely despite the 421.8 bit/s rate of the broadcast channel.

3.1 Assumptions and threat model

RevCast's design is driven by several modest assumptions about the rate at which CAs revoke certificates. We assume that, *over a very small window of time* (on the order of 10 seconds), the following hold: (1) any given CA revokes few certificates (typically none, but often fewer than a dozen), and (2) at most only a few CAs issue revocations. We emphasize that these assumptions only apply to these very small windows of time, not across, say, hours or days. In Section 5, we demonstrate, with a measurement study of 621 CAs over two months, that these assumptions do hold in practice.

When operating at such small time windows, the primary challenge lies not in delivering the revocations, but in delivering CAs' attestations that *nothing was revoked* since the last message. These attestations are ever-present, they come from almost all CAs, and they must be delivered at a very high rate for users to ensure that their revocation state is up-to-date.

RevCast's threat model mirrors that of today's PKI: end-users can make their own trust assumptions regarding individual CAs. We note that RevCast does not require additional trust assumptions regarding the FM tower. Towers may arbitrarily drop, alter, or repeat transmissions from an arbitrary subset of CAs. Moreover, attackers may jam or drown-out FM towers with their own broadcasts. As we will demonstrate in this section, RevCast remains secure in the presence of such untrusted communication because it uses FM merely as a means of dissemination—authentication is achieved end-to-end (between CA and end-user).

3.2 A single CA

To simplify the presentation, let us first consider a single CA making sole use of a tower to transmit its revocation state. The critical information involved in a revocation r—that is, what is required in X.509 [6]—consists of[1]:

- The CA's identity and public key identifier.
- The serial number, s, of the revoked certificate.
- A timestamp, t, signifying at what time the certificate should be considered revoked.
- The CA's signature, σ, of the above information.

CA identifier.

Note that the CA's identity and key change at a much slower rate than the CA's revocation state. Rather than repeatedly broadcast these identifiers, the tower maintains a simple web server that receivers can go to very infrequently to learn of the associated CA's identifier and key.

Revocations.

Suppose that our given CA generates revocation r_i at time t_i. To put timing into perspective, note that a 4096 bit signature would take 9.71 seconds to transmit over our 421.8 bit/s RDS link. Our analysis of a large CRL dataset (Section 5.3) indicates that the time between consecutive revocations, $t_{i+1} - t_i$, is typically greater than 10 seconds (leaving enough time for the largest signatures we have observed), but not always. In the event that $t_{i+1} - t_i$ is less than the time to transmit r_i's revocation data, we could face unnecessary delays were we to simply stream revocations as they occur from the CA to the tower. Were the tower to simply wait and deliver a signature across both revocations r_i and r_{i+1}, this delay could have been avoided.

Fortunately, the long time to broadcast revocation data provides a natural *window* of time during which to aggregate revocation data. Suppose that the tower can send a new transmission every w seconds. The CA collates all of its revocation data over w seconds, resulting in a (typically empty) set of revoked, time-tagged serial numbers $S = \{(t_1, s_1), \ldots, (t_k, s_k)\}$. Nearing the end of a broadcast window, if S is non-empty, then the CA sends to the tower the set S, and a signature over the entire set (we discuss empty sets next).

One alternative to this fixed parameter w would be to simply wait until the previous transmission is complete. However, CAs sometimes (though quite rarely) appear to deliver many revocations all at once (Section 5.3). Allowing the set of revoked serial numbers S to grow unbounded would risk having end hosts become arbitrarily inconsistent with the tower. Instead, when S is too large to be able to transmit within time w, then the CA sends a message attesting to the fact that there are too many new revocations to broadcast. It can either split these up into successive broadcast windows or, in the extreme case, simply suggest to end hosts that they directly obtain the updated CRL.

Nothing-now messages.

Over a short period of time, it is likely that a given CA has no revocation to send (Section 5.3). Were we to broadcast nothing during these times, then end hosts would have no way to distinguish between a period of no revocations and, say, a malicious tower who has failed to forward the CA's revocation state. For end hosts to know that their revocation information is up to date, a CA must provide "nothing-now" messages: signed attestations that there have been no additional revocations since the last broadcast window.

More concretely, at the beginning of a broadcast window, the tower delivers to the CA a timestamp T denoting a time near the end of the broadcast window (roughly w seconds into the future). If the CA has made no revocations by time T, then it signs the message $\perp \; ||w||T$, and sends the signature to the tower (in practice, $\perp$ could be a constant, well-known string, e.g., "nothing"). This represents a signed attestation that "there have been no revocations in the interval $[T - w, T]$." Thus, at every time interval, the CA sends one of two signed messages: either a list of revoked certificates' serial numbers, or a message indicating that there are no updates.

3.3 Multiple CAs

While a single CA could vastly improve timely delivery of revocation state for a large fraction of the web's certificates (two CAs from our dataset each account for roughly 39% of all of the revocations), we extend our protocol to allow many more CAs to make simultaneous use of a tower. In the remainder of this section, suppose there are N CAs who all wish to make simultaneous use of a given tower.

A strawman approach would be to simply append all N CAs' revocation state. This would result in broadcasts that

[1]There are various extensions that include information such as the reason for the revocation, but since those are not strictly necessary to determine if a certificate is revoked, we do not consider them here.

consist of N signatures and up to N sets of revoked data. That is, the messages would typically be roughly N times larger than in the single-CA case, and thus would decrease the timeliness of the information by a factor of roughly N, as well. In the event that all N CAs have new revocations to issue within a given broadcast window, this may be the best that we can achieve.

Revocations.

However, as our results in Section 5.3 show, it is extremely uncommon for more than two CAs to have revocations to send within a broadcast window of ten or less seconds. We can therefore expect that there are typically only a small, constant number of revocations from an even smaller set of CAs to send within any broadcast. As a result, we append these messages, typically resulting in at most two signatures (and often none).

We compactly represent the originator of the individual revocations by maintaining a simple lookup table at the tower. This table maps CA c_i, for $0 \leq i \leq N-1$, to its full CA identifier and key identifier strings. We expect that this set of CAs changes very infrequently, on the order of months—in practice, the tower would likely have a business agreement with a CA to transmit its revocation information. Receivers must obtain this list whenever it updates, but doing so allows the tower to only transmit the index i of the CA, instead of the much longer identifier strings.

As a result, an individual revocation appears on the broadcast medium as $i||S_i||\sigma_i$, comprising the revoking CA (i), the time-stamped, revoked certificates serial numbers $S_i = \{(t_{i,1}, s_{i,1}), \ldots, (t_{i,k}, s_{i,k})\}$, and signature σ_i.

Saving bandwidth with multi-signatures.

The remaining challenge is to represent all of the nothing-now signatures that all the other CAs must include. Recall that a nothing-now message allows a CA to attest that it has no new revocations since the last broadcast window.

Our key insight is that all CAs with nothing to send in a given broadcast window can all sign the same message ($\perp$, along with a recent, tower-supplied timestamp). We can thus make use of a *multi-signature* [12] scheme, which allows a set of N signers to sign the same message, resulting in a single, "compact" signature with size equal to the size of the largest signing key. Compared to the more general "aggregate signature"—which allows a single signature over *multiple* messages—multi-signatures can exploit the assumption of a common message to achieve faster verification times.

We provide a background on multi-signatures using bilinear maps, and then describe our protocol which uses them.

Background: multi-signatures with bilinear maps.

Recall that a bilinear map consists of a function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$, with the special property that for any generator $g \in \mathbb{G}$, $e(g^a, g^b) = e(g, g)^{ab}$. For groups $\mathbb{G}$ over which the discrete log problem is hard (that is, given g and g^a, it is hard to determine a), bilinear maps allow for signature schemes, wherein the private key is some random a and the public key is g^a for a publicly known generator $g \in \mathbb{G}$. Boneh, Lynn, and Shacham [5] described an elegant construction of signatures in which signing a message m consists of simply computing $H(m)^a$; verification consists of ensuring that $e(H(m)^a, g) = e(H(m), g^a)$.

Boldyreva [4] demonstrated that bilinear map-based schemes also permit a straightforward protocol for multi-signatures; we review this briefly here. Suppose, for simplicity, that two parties with private keys a and b wish to sign the same message m. In essence, a multi-signature results in a new secret key $a + b$; the signed message is $H(m)^a \cdot H(m)^b = H(m)^{a+b}$. Note that this can be computed *nonsequentially*: the two signers could perform their action in parallel, and either of them (or a third party) could combine them into the multi-signature. Verification consists of ensuring that $e(H(m)^{a+b}, g) = e(H(m), g^{a+b})$. Note that g^{a+b} can be computed by simply multiplying the two signers' public keys together: an operation that any verifier could perform.

One of the main attacks that significantly complicates multi-signatures in most settings is known as a *rogue key attack* [24]. The attack proceeds as follows: Given a victim's public key g^a, although it is difficult to compute a, it is easy to invert this number to obtain g^{-a}. An attacker can thus, without knowing a, assert ownership of a public key $g^r \cdot g^{-a} = g^{r-a}$ for some randomly chosen r. Thus, a message signed with r, i.e., $H(m)^r$, would appear to be signed by both a and r, because the equivalent multi-signature public key would be $g^{a+(r-a)} = g^r$.

Rogue key attacks are possible when the attacker can assert that he owns public key g^{r-a} without having to prove it. Ristenpart and Yilek [24] showed that they can be avoided by requiring simple proofs of possession, that is, by requiring that the attacker sign a message using his public key g^{r-a}. This requirement is often considered unreasonable in today's PKI, as it would require modifications to existing key sharing schemes. However, as we show next, our application permits including these proofs of possession in-band.

Nothing-now messages.

Using Boldyreva's multi-signature scheme, our protocol proceeds as follows. Suppose CA c_i has key pair $(\mathsf{PK}_i, \mathsf{SK}_i) = (a, g^a)$ for some well-known g and random a, as above. At the start of a broadcast window, the tower sends to each CA a timestamp T representing when the next broadcast will be sent. Each CA first verifies that this is a valid timestamp approximately w seconds (the length of the broadcast window) into the future. Nearing the end of the broadcast window, suppose the set of CAs who have no revocations to report is R. Each CA $c_i \in R$ computes signature $\sigma_{\perp,i} = H(\perp ||w||T)^{\mathsf{SK}_i}$, and sends $\sigma_{\perp,i}$ to the tower.

We can avoid the rogue key attack in our setting by requiring that, before entering a multi-signature, each CA must have issued a verifiable *individual* signature—a revocation—before entering a multi-signature. This individual signature serves as proof that the sender owns his public key, and can thus be safely assumed not to be launching a rogue key attack in our multi-signature scheme. Such a proof of possession also requires that RevCast use different hash functions for individual signatures and nothing-now signatures [24] (in practice, this can be achieved by using the same hash function, but simply with different prefixes, $H(0||x)$ and $H(1||x)$).

An honest tower may refuse to include signatures in its broadcast for two reasons. First, the tower does not include signatures that it cannot verify; to do otherwise would allow a single signer to make the multi-signature fail to verify for all parties. Second, if the tower does not receive CA c_i's message before its scheduled transmission time T, then the tower

proceeds without c_i's signature that round; to do otherwise would allow a lazy (or malicious) CA to delay everyone's revocation state from being sent. It is thus important that each CA remain online and available, which is consistent with the assumptions of CA availability today. Suppose w.l.o.g. that each CA $c_i \in R$ returns its signature by the deadline, and that the tower is able to successfully verify all of them.

Having collected and verified signatures from the CAs in R, the tower constructs the multi-signature by simply multiplying the signatures together: $\sigma_\perp = \prod_{i \in R} \sigma_{\perp,i}$. As a final step, the tower broadcasts: (1) individual revocations (if any), (2) the timestamp T, and (3) the multi-signature $\sigma_\perp$.

Receivers verify the individual revocations as in the single-CA case. To verify the multi-signature $\sigma_\perp$, the receiver reconstructs R—note that this is simply the set of all CAs registered at the tower minus those who issued individual revocations in this time interval—and checks that $e(\sigma_\perp, g) = e(H(\perp ||w||T), \prod_{i \in R} \mathsf{PK}_i)$.

3.4 Handling receiver loss

Our protocol thus far has assumed that every receiver obtains all transmissions, but as our FM RDS measurement results demonstrate (Section 4), receivers near the periphery of a tower's service contour can experience low but consistent loss. We extend our protocol once more to assist these users, keeping them from having to query the CAs repeatedly.

Our insight is to exploit the fact that, within most broadcast windows, there are no revocations from any CA. If a receiver fails to obtain the data from one or more consecutive broadcast windows, we seek a means by which that receiver could ascertain whether it has missed any updates.

To this end, CAs optionally participate in one more multi-signature. This "nothing-since" attestation is meant to represent says that none of the set of signing CAs have issued any updates within the last W seconds: a larger window of time than our broadcast windows ($W > w$). This follows a nearly identical protocol to the nothing-now multi-signatures: In addition to the timestamp of the next broadcast, T, the tower also sends some time in the past, $T - W$. If the CA has issued no revocations within the time period $[T-W, T]$, then it signs the message $\perp ||W||T$. As before, the tower collates the signatures by multiplying them together. Note that this is a strict generalization of the nothing-now messages, which are nothing-since messages with $W = w$.

In addition to sending the multi-signature, the tower must also compactly represent the set C of CAs who are included in the nothing-since signature. This was not necessary for nothing-now messages, since the set of CAs who have nothing to send now is simply the set of all CAs set-minus the set of CAs who issued individual signatures. We represent C with a bit array of length N, and transmit either the bit array itself or in run-length encoded form, whichever is smaller for that transmission.

The extension as described above introduces a redundancy: any CA providing a nothing-since signature implicitly has no revocations now, yet in our scheme it also provides a nothing-now signature. This permits a straightforward optimization wherein each CA signs at most one message: a revocation, a nothing-since (if it has not revoked in the last W seconds), or a nothing-now (if it is not revoking now, but has revoked within the past W seconds). Receivers infer the set of CAs involved in the nothing-now multi-signature by simply removing both the CAs who provided individual signatures as well as those who signed the nothing-since message.

We note in closing that, were our protocol applied to a more reliable broadcast medium, nothing-since messages may not be strictly necessary, though it would allow receivers to occasionally enter a power-save state.

3.5 A day in the life of a receiver

We summarize our design by discussing how a receiver interacts with our system. If the receiver has not obtained the tower's table mapping index i ($0 \leq i \leq N-1$) to each CA c_i's identity and key identifier strings, then it obtains this as well as tower-specified parameters w and W by accessing, say, a web server run by the tower.

The receiver then tunes to the station and, within each receiver window, obtains a message that comprises the following components:

- Individual revocations, $i||t_1||s_1|| \cdots ||t_k||s_k||\sigma_i$, where i represents the tower-supplied index for CA c_i, the (t, s) pairs represent the timestamp and serial number of certificates that c_i revokes, and σ_i is c_i's signature over this list.
- A list of CAs who failed to reply to the tower in time for inclusion in the broadcast.
- Nothing-now multi-signatures, $T||\sigma_\perp$, where T represents a tower-supplied timestamp and $\sigma_\perp$ is the multi-signature signed by all CAs except those who issued individual revocations or failed to reply in time.
- Nothing-since multi-signatures $C||\sigma'_\perp$, where C represents the list of CAs who have signed multi-signature $\sigma'_\perp$, asserting that they have not issued any revocations in the period $[T - W, T]$.

If any of the above signatures fail verification, then the receiver drops the corresponding data, and treats it as if he failed to obtain the broadcast.

Suppose the receiver missed the previous m broadcast windows, that is, that his state is $m * w$ seconds old. If $m * w \leq W$, then the receiver can determine (from the nothing-since message) from whom he has missed prior revocations, and request them directly from the corresponding CAs, e.g., by downloading their CRLs. If, on the other hand, $m * w > W$, then the receiver has missed too many messages for the nothing-since messages to help—perhaps the receiver just turned on for the first time, or experienced a prolonged failure. In such an event, the receiver must obtain, one-time, the CRLs from each of the CAs it trusts, but can then use any broadcasts it obtains in the interim.

Note that throughout this process, the tower is merely a forwarder: though the tower could potentially be an active attacker, the results of Boldyreva's multi-signature scheme ensure that the end-user can still verify the authenticity and non-repudiability of each CA's message. As such, the security analysis follows as a direct application of Boldyreva's scheme.

3.6 RSA-based multi-signatures instead?

Our protocol makes use of bilinear maps over elliptic curves, but in practice today, CAs almost solely use RSA. Ideally, we would make use only of the RSA key pairs that CAs already have. Here, we briefly describe our rationale behind

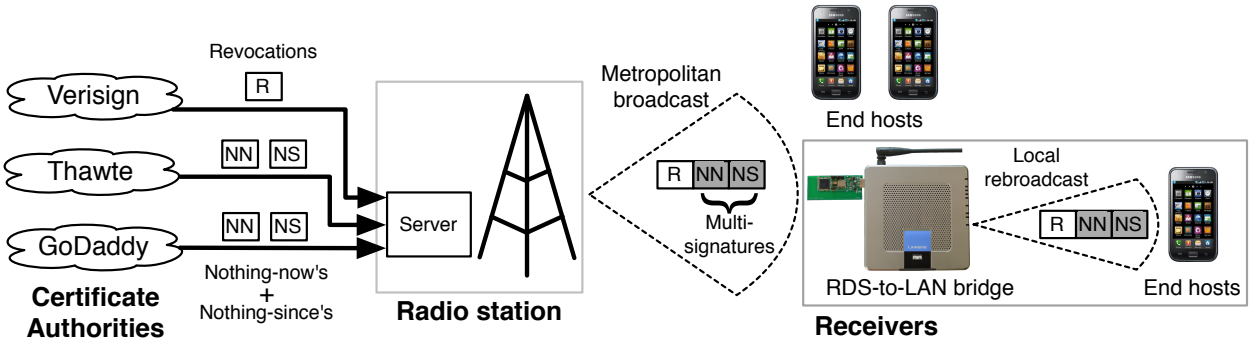

Figure 1: Overview of RevCast.

choosing ECC, and potential areas of future work to more readily permit an RSA-based approach.

Adopting use of our system would require that CAs generate elliptic curve keys (a, g^a) under some well-known generator g, and to attest to them. This process is possible incrementally and without requiring additional trust assumptions on behalf of users. One such construction could proceed by having a CA sign its public key g^a with its existing RSA public key, and disseminating this info (e.g., directly off of the CA's website and through the tower itself). Because the trust in the elliptic curve-based key would be rooted in the RSA key, clients' root CA information need not be modified.

Elliptic curve techniques have several benefits over their RSA-based counterparts, most notably smaller key sizes (and thus less bandwidth consumed). Additionally, with respect to multi-signature schemes, the bilinear map-based construction we use [4] has the attractive property that it does not require sequential signing. All RSA-based multi-signature schemes of which we are aware require sequential signing or interaction; a non-sequential RSA-based multi-signature scheme would be extremely powerful in our setting. Nonetheless, given the duration of our broadcast windows, sequential signing may be feasible in practice for a moderately sized set of CAs.

Finally, the bilinear maps-based scheme we use is, we have found, extremely simple to express and to implement. Other RSA-based schemes of which we are aware are considerably more complex [20], but it is possible that this would be improved in the future.

4. DESIGN AND IMPLEMENTATION

In this section, we describe the design and prototype implementation of a full end-to-end RevCast system that makes use of the transmission protocol from Section 3. Figure 1 illustrates an overview: RevCast transmissions come from CAs to an Internet-connected server in the radio station. End hosts can receive RevCast directly through an embedded FM RDS receiver, or via a proximal FM RDS receiver that locally rebroadcasts (e.g., over a wireless access point) messages to clients that missed broadcasts while powered-off and, for incremental deployment, to non-FM RDS enabled clients.

4.1 Transmitter

Installed at the radio station, the RevCast transmitter software runs on an Internet-connected computer that can send data to the RDS encoder with Ethernet or serial connection. This is in line with the current server deployments at radio stations to download and broadcast artist and song information over RDS. Additionally, our transmitter can authenticate, authorize, and charge CAs to transmit their revocations, but this is outside of the scope of this paper.

The underlying RDS ODA protocol provides nominal headers, but to support our variable-sized transmissions, we have constructed our own framing scheme. We call each unit of transmission a *package*, which consists of *segments*, which are in turn (sender(s), data, signature(s)) triples. The beginning of a package, the segments within a package, and the signature are delimited. This is driven by the principle that a radio's wake-up time varies and a receiver should be able to detect the start of a new segment they want to receive. We use the RDS B group as a delimiter (16 modifiable bits), followed by the data in RDS A groups (37 modifiable bits).

4.2 Receiver

A RevCast receiver is simply any client that can obtain and verify transmissions from our over-the-air protocol. As a means to bootstrap deployment to existing devices, we have developed a last-mile link receiver prototype for PCs and OpenWRT routers that receives broadcasts and optionally rebroadcasts them over a local wireless network. In this section we describe the receiver hardware prototype, the corresponding client-side software, and how the power consumption of a RevCast receiver will not significantly drain the batteries of RDS-enabled smartphones.

RDS-to-LAN bridge.

To realize our vision of a pervasive deployment of RevCast receivers in Internet-connected devices, we designed a *RDS-to-LAN bridge*: a low-cost, low-power RDS receiver that can be plugged into residential wireless access points via USB, over which it can then rebroadcast revocation state. We envision the RDS-to-LAN bridge deployment in routers as a last-mile link. Our RDS-to-LAN bridge prototype includes a 1/30th wavelength monopole antenna trace on the PCB, and a Silicon Labs si4705 FM receiver integrated circuit. These chipsets have several nice properties that make them ideal for a pervasive deployment: they are inexpensive, small (the si4705 is 3 mm^2), power-friendly, and they require only a few external components.

Client-side API.

We implemented a receiver in C that runs on OpenWRT routers, PCs, and smartphones. Our design takes advantage

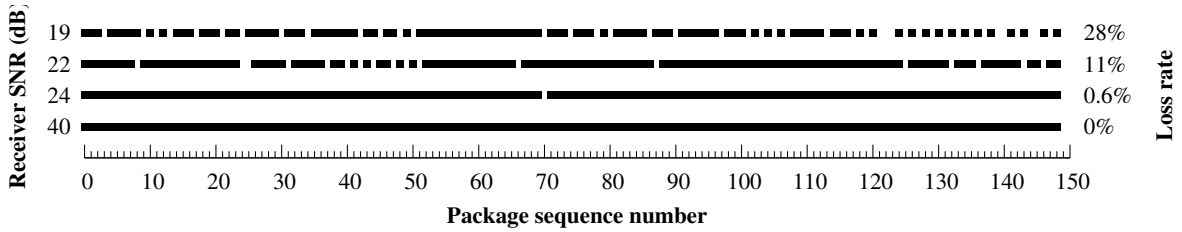

Figure 2: Broadcasting 149 packages (x-axis), each 160 bytes, from a 3 kW tower to four RevCast RDS-to-LAN bridges (y-axis). A black box indicates the package's signature verified, a white box indicates either the signature did not verify or there were missing RDS groups. RDS error correction is disabled. The high SNR receivers capture >99% of the packages. The low SNR receivers miss many, but not all, and they only miss three of the same packages.

of the continuous-stream nature of broadcasts; as there is always incoming data, a receiver can easily detect that it missed a transmission. The client can receive broadcasts from a local receiver (as is the case with an RDS-enabled smartphone) or over LAN multicast from an RDS-to-LAN bridge. The client-side API consists of three methods:

- **Subscription**: Applications ask the receiver to subscribe to RevCast segments on a given tower. The receiver then tunes to the specified tower, and stores on all incoming RevCast-related transmission on its local, MicroSD storage. The receiver's subscription service is a basic TCP server that takes subscriptions as input and outputs segments to applications.
- **Local-retrieval**: Clients may not be able (or want) to stay powered-on to receive all broadcasts. Instead, while powered-off, the local RDS-to-LAN bridge stores incoming broadcasts, which the client can explicitly retrieve after powering back on. We have embedded a 16 GB MicroSD card in our receiver, which allows for storage of ten years of continuous broadcasts. We run a small TCP server on the receiver, that allows clients to obtain the segments reliably and in order.
- **Verification**: Finally, upon retrieving all of the incoming revocation state, the client verifies the relevant signatures, and discards any that fail verification. A client could perform verification of all incoming signatures, but a reasonable optimization would be to immediately verify revocation signatures only if the client holds the corresponding certificate.

Power consumption.

Because RevCast runs continuously on all devices, it is important that receiving and verifying revocations from RevCast remain power-friendly. Here, we describe why RevCast will not significantly drain the battery of an RDS-enabled smartphone. The primary reason is that, in the common case (when there are no new revocations), RevCast will not require continuous operation of the RDS receiver. With 621 CAs, RevCast requires 2.89 sec to receive the package comprised of nothing-now and nothing-since messages. The receiver could then be powered off for the remainder of the broadcast window (tens of seconds in our experiments), minus the time it takes the receiver to turn on and tune in. As a concrete example, we measured the si4705 chipset to take ∼0.7 sec to turn on and tune in. Moreover, the si4705 can receive most of the 2.89 seconds-long package without having to power on the smartphone's CPU because the si4705 has an internal FIFO that can store up to 2.19 seconds worth of RDS messages.

Location	SNR	Dist. from tower
Apartment building	40	0.85 km
Apartment building	24	5.5 km
University building interior	22	2.7 km
Apartment building	19	8.0 km

Table 1: RevCast prototype receiver locations in the testbed deployment

Using the Google Nexus 5's power consumption as an example, we can compare the energy consumed receiving revocations with RevCast to normal operation. To power on, tune in, and receive a package consisting only of nothing-now and nothing-since messages takes the si4705 3.59 sec. The si4705's power consumption while receiving RDS is only 64 mW. This results in a total consumption of 230 mJ or the equivalent of 605 msec of idle CPU and dimmest screen brightness (which consumes 380 mW). RevCast will also consume some energy to store the package until the smartphone needs it. Assuming 20 sec broadcast windows, continuous reception of RevCast on a Google Nexus 5 for an entire day would consume as much power as leaving the phone idle, and with dimmed screen, for 0.73 hours. This is a reasonable cost for most users, but further optimizations are possible.

4.3 Metropolitan micro-benchmarks

We evaluate RevCast's transmitter and receiver prototypes in a metropolitan area with a 3 kW commercial FM radio station located in the center of a mid-sized metropolitan area and four RDS-to-LAN bridges.

The station agreed to let us send RevCast's RDS messages 50% of the time with their artist and track title RDS messages occupying the other 50%. Because RevCast transmits with a different RDS message identifier than the station, it should not interfere, and we are not aware of any complaints of radio problems from listeners of the station.

We transmitted 149 RevCast 160 byte packages simultaneously to the four receivers and observed if the signatures verified successfully. Figure 2 shows the results. We placed two receivers in a high signal strength area, and two in a low signal strength area, as shown in Table 1. The high signal strength receivers received over 99% of the packages. The

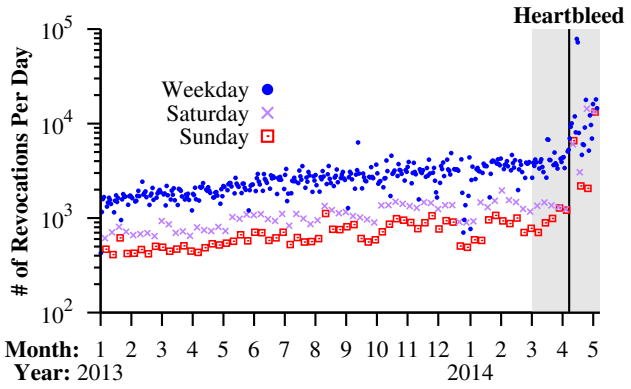

Figure 3: Number of revocations per day between January 2013 and April 2014. We analyze the highlighted time period, the month before and after Heartbleed, in this study.

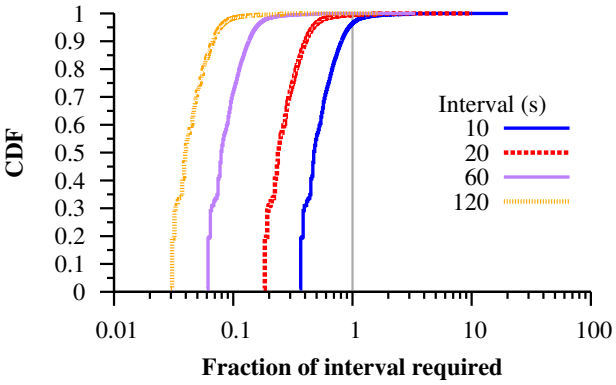

Figure 4: Even while transmitting revocations within 10 seconds, RevCast requires significantly less than the resources available in FM RDS for 96% of the intervals where there are revocations.

24 SNR receiver operates on the edge of this high reception probability region, and it only had errors in one package.

An even distribution of RDS group errors would cause the receivers to lose many, if not all, of the 160 byte packages (each 160 byte package is made up of ∼35 RDS groups). The lowest signal strength receiver was able to receive 72% of the 149 transmitted packages. This is an encouraging result. It indicates that RDS receivers with small trace antennas can receive packages, albeit with a moderate package loss rate, when they are a few kilometers away from a moderately powered transmitter.

5. TRACE-BASED EVALUATION

In this section, we feed a stream of 114,021 certificate revocations through a simulation of RevCast, subject to the FM loss model measured in Section 4.3. Our results show that: (1) 96% of the time RevCast can transmit all revocations within 10 seconds, (2) the nothing-since attestations are remarkably effective at eliminating CRL checks, and (3) when there were extreme revocation rates as a result of the widespread Heartbleed vulnerability [30], 70% of the time RevCast could transmit all revocations within 10 seconds. Further, we reinforce the validity of the trace-driven simulation by studying the CA revocation behavior our dataset.

5.1 CRL dataset

The dataset of certificate revocations that we analyze were collected by Zhang et al. [30] from the CRLs for public-facing SSL servers on the Internet. Rapid7[2] collected SSL certificates from the entire IPv4 address space from October 2013 to April 2014. Zhang et al. extracted the CRL extension field from these certificates; then they downloaded the CRLs on May 6, 2014. There are 974 unique CRLs in this dataset from 621 CAs, totaling 87 MB.

Figure 3 shows the number of revocations per day in the dataset. On a typical day, the number of revocations per day is on the order of thousands: even a low bandwidth distribution system may be able to keep up with the rate of revocations. The number of revocations is significantly lower on weekends compared to weekdays (GMT). Interestingly, the number of revocations on Saturdays tends to outweigh Sundays, except in the wake of Heartbleed. The number of revocations in each hour also exhibits a diurnal pattern, confirming that many certificate revocations are issued during business hours. This could be explained by the fact that most revocations are likely benign (e.g., a CA issuing a certificate to supersede one that is nearing expiry), though Zhang et al. also find that even compromised certificates are typically only revoked during traditional business hours [30].

Over time, CAs can remove revocations from their CRLs for at least three reasons (there may be others that are not documented): (1) The RFC requires that revocations must be removed from CRLs one CRL issue period after the certificate expires [6]. (2) Revocations with the reason code that the certificate is on hold may be temporary revocations. (3) Revocations may only exist in a CRL for an operator specified amount of time. As such, we constrain our dataset to only the month prior to Heartbleed and the month following Heartbleed (March, 2014 to April 2014).

Our resulting, two-month-long dataset which we use in the remainder of our evaluation comprises 114,021 revocations before Heartbleed and 402,747 revocations after Heartbleed. The gray box in Figure 3 indicates the days covered by this time period.

5.2 RevCast simulation results

Using a simulation, we begin our evaluation by measuring how well the small protocol messages in RevCast can distribute 114,021 revocations over the one-month period before Heartbleed. We are interested in two features: how many intervals include revocations and how often must a client resort to pulling the CRL due to loss when using the nothing-since attestation. We vary the length of the interval: a shorter interval provides faster revocation but limits the number of revocations that can be transmitted in that interval.

We simulate transmitting all revocations over the one-month period with all of the fields in the RevCast protocol. This includes the three segments in each package as well as sizes and delimiters for framing so the receivers can parse

[2]https://scans.io/study/sonar.ssl

No. Since	Loss Rate	Pulls / Avoided	Pull Int. (hr)
1 min	0.6%	425 / 109,257	1.66
	11%	8,067 / 1,953,529	0.07
	28%	19,463 / 4,691,474	0.03
2 min	0.6%	1,500 / 242,590	0.83
	11%	27,031 / 4,293,996	0.03
	28%	64,928 / 10,298,436	0.02
3 min	0.6%	2,380 / 291,796	0.83
	11%	42,079 / 5,129,459	0.03
	28%	101,353 / 12,298,736	0.01

Table 2: The attested nothing-since allows receivers with moderate losses to avoid pulling many CRLs to make up for the losses.

the messages out of a continuous RDS broadcast. We assume 233-bit signatures. When there are are no revocations, the protocol requires at most 2.89 seconds to transmit the package.

Figure 4 shows, for broadcast windows of 10, 20, 60, and 120 seconds, what fraction of the interval is used to transmit all necessary RDS data. Only 3.7% of the 10-second intervals included too many certificate revocations from too many CAs to be completely conveyed. A two-minute interval is long enough to transmit 99.995% of the revocations that occur during any two-minute interval in the data. With coordination with the CAs, there may be a means to smooth any bursts in revocations so that they are spread across shorter intervals, which we leave to future work.

We simulate transmitting to receivers that have some losses. Losses in FM may occur because receivers are outside the main coverage area of the transmitter or because they are moving. Such receivers will usually receive revocations as transmitted, but when there are losses, they may need to pull a CRL over the Internet. The nothing-since segment (Section 3.4) allows receivers to know which CAs had nothing to revoke over an interval. This consumes one bit per CA (621 bits), though it could be more compactly transmitted with run length encoding.

The loss experiment is trace-driven by the loss patterns observed in the metropolitan micro-benchmark in Section 4.3. Because the pattern of losses is particular to FM RDS (losses are not uniform), we repeat the loss patterns over the one month period. We ran it on the three receivers which had low enough SNR from the tower that they experienced some loss: 0.6%, 11%, and 28%. The results in Table 2 indicate, for one, two, and three minute nothing-since messages, how many CRL pulls were necessary, how many pulls were avoided during losses due to the existence of nothing-since messages, and the median interval between pulls. The results indicate that a receiver in a position such that it has a low loss rate (e.g., 0.6%) can receive revocations while only requiring hundreds of delta CRL pulls over a one month period. The results also demonstrate the power of nothing-since messages: although the receivers that experience significant losses must pull more often, the nothing-since messages allow them to avoid over 99% of the pulls they would have otherwise needed.

One remaining question is: are losses correlated across many receivers? If so, they might flood CRLs with pull requests when there is a loss. The micro-benchmark results in Section 4.3 indicate that this is unlikely as the 11% and 28% loss rate receivers only missed three of the same packages. If the losses were independent, we would expect the two receivers to lose five of the same packages. Independent loss is expected because losses are likely to be due to a receiver independently losing synchronization with the transmitted signal rather than due to a widespread property of the transmission.

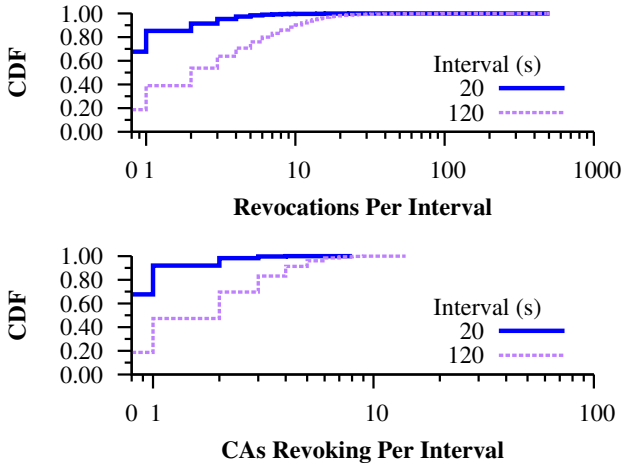

Figure 5: Within any given 20 or 120 second window of time, few CAs issue any revocations; moreover, when there are revocations within such small windows, they usually come from only one or two distinct CAs.

5.3 Why RevCast works

The simulation results show that RevCast is viable for distributing revocations, but the remaining question is: what is the nature of certificate revocations that makes RevCast work so well in practice?

Many revocations in the same interval are uncommon.

Recall from Figure 4 that RevCast is able to deliver almost all revocation state within broadcast windows of ten or more seconds. Figure 5 (top) demonstrates that this is in large part due to the fact that, within these small windows of time, there are few revocations to send. Moreover, when there are individual revocations to send, they usually come from one or two CAs; as a result, RevCast is typically able to represent individual revocations using at most three signatures.

Most of the time, no CAs have any revocations to send. RevCast excels in these time windows through its use of multi-signatures for the nothing-now messages. As a result, users learn that all N CAs have nothing to revoke at the cost of a single, compact signature.

Most CAs revoke infrequently.

The nothing-since field keeps the number of CRL pulls low when there are losses. The nothing-since field works well when CAs revoke sparsely. Figure 5 (bottom) shows the distribution of the number of CAs issuing revocations within a given 20- or 120-second interval over the pre-Heartbleed month. In the majority of all 120-second intervals over the course of a month, there are two or fewer CAs revoking in

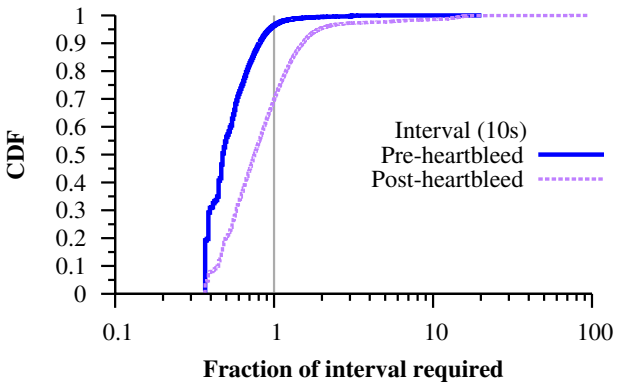

Figure 6: RevCast's ability to handle revocation rates in typical and in disastrous settings.

that interval. For 90% of 20-second intervals, there is at most one CA who issues a revocation. This demonstrates the importance, and the power, of our nothing-since messages.

5.4 Worst-case scenario: Heartbleed

These observations—that in small windows, CAs do not revoke much—holds in the normal case, but not always. In early April 2014, a buffer over-run vulnerability in OpenSSL, called Heartbleed [28, 9], risked exposing servers' private keys, resulting in a nearly two orders of magnitude increase in the rate of revocations (Fig. 3). We ran our trace-based evaluations using a month of revocations starting from the day that Heartbleed was publicly announced, April 7. Figure 6 demonstrates that for 70% of 10-second intervals, RevCast was able to deliver all revocation state without any additional delays (compare this to the 96% for the typical case). 30% of the time, RevCast required additional time to handle the uncharacteristically large influx, though typically less than a minute. In the extreme case, RevCast required 15.5 minutes to clear its buffer of revocations.

While this number may seem high, it is crucial to note what this represents: in the wake of a disastrous vulnerability, RevCast was able to deliver *all* revocations within 15.5 minutes of when the revocation was first announced. Certainly, had a standard browser pulled immediately at this time, it could have downloaded the revocations more quickly, but in practice browsers do so on the order of hours or days. As such, this is a vast improvement over the revocation systems of today, even in our worst-case scenario.

6. RELATED WORK

Revocation dissemination ideally satisfies three properties: timeliness, low cost of operation, and privacy. There have been many schemes that have focused on various combinations of these properties. Recall that, traditionally, a CA hosts its CRLs, which end hosts download directly. Because end hosts download the entire list of revocations, this approach protects user privacy. However, this simple pull-everything scheme comes at the cost of transmitting so much data (the entire list of revoked certificates) that it would not be possible to simultaneously satisfy timeliness for all users will maintaining low cost of deployment.

The Online Certificate Status Protocol (OCSP) [26] addresses these high costs of deployment by allowing users to query for the status of a certificate in particular. This decreases the amount of information a user has to pull, but at the cost of exposing users' browsing habits. Nonetheless, the decrease in cost has led to a fairly widespread, mainstream deployment of OCSP.

OCSP stapling [21] and short-lived certs [29] are modifications to existing revocation systems to improve privacy and reduce bandwidth required by CAs, but both fundamentally require far more bandwidth from the CA to disseminate revocation as quickly as RevCast. The bandwidth consumed by both OCSP stapling and short-lived certs increases with the number of active certificates. RevCast takes advantage of the scalability and privacy that broadcast naturally provides, so the CAs' bandwidth consumption increases only with the number of revocations (and broadcast windows, e.g., 10 sec intervals, in a day). Chrome CRLSets [11] require little bandwidth from the CAs, but the bandwidth required from Google increases with the number of Chrome installs; this is likely why their revocation update interval is so long.

Many schemes have been proposed that broadly follow the design where a CA sends to a set of distributed "directories" that users subsequently query. Most such approaches seek to lower the raw amount of bandwidth from CA to directory and from directory to user [13, 10, 19, 17, 8, 1], while others seek to support more sophisticated trust models than today's PKI supports [31].

Fundamentally, all of these approaches are faced with the same trade-offs of timeliness versus bandwidth, and thus they typically assume updates come infrequently (once a day). A key technical challenge many of them face is making it easy to verify that a certificate has *not* been revoked without requiring a user to download the entire revocation list with every query [17, 8, 1]. Because we can operate in a world where everyone gets everything, many of these issues become trivial: a user knows that a given CA has not revoked certificate c if the user has obtained all of the CA's attestations, and if none of them include the individual revocation of c. Our key technical challenge lies elsewhere: in making sure that all users can get all CAs' data despite having a very narrow broadcast bandwidth in which to operate.

We are not the first to propose using FM RDS as a generic link for Internet applications. Rahmati et al. [22] demonstrate that it is feasible to construct large, repairable messages out of RDS's eight byte messages. They also describe some of the higher-layer challenges in deploying a general data RDS broadcast system on existing FM radio stations. RevCast represents a more concrete application of the FM RDS medium; we view it as complementary to this work.

Finally, AlertFM [2] also uses RDS via leased bandwidth to distribute local and national emergency alerts to receivers of their own design, including both standalone text displays and USB based for windows PCs. We see the existence of AlertFM as a demonstration that it is feasible to inject timely RDS at scale, and its obscurity as another sign that the market is not enthusiastic about special purpose receivers. Because RevCast can operate over any existing FM RDS receiver, we are optimistic that it has a higher likelihood of adoption.

7. CONCLUSION

We have presented RevCast, a revocation dissemination system built upon FM radio broadcast. RevCast demonstrates that, with a new communication primitive—metropolitan-area broadcast—it is possible to simultaneously achieve timeliness and privacy at low cost. RevCast thereby challenges the established belief that these three properties face fundamental trade-offs. RevCast achieves this despite FM RDS's 421.8 bps bitrate by applying a compact multi-signature scheme [4].

One of the main contributions of RevCast is the observation that radio broadcast can be highly practical because: (1) small receiver hardware exists and is even already installed in most smartphones (2) tiny FM antennas can be integrated into phones because the spectrum used by FM stations is only 200 kHz (3) RDS is an open standard and there are no licenses for the receivers or transmitters and (4) the radio stations we spoke to were willing to share their RDS bandwidth with us; so radio stations have RDS bandwidth to spare. Although it is difficult to predict what systems are likely to experience wide-scale deployment, these observations indicate that fast, private revocation over FM is surprisingly practical.

There are several interesting avenues of future work. Our evaluation focuses on the question of how much revocation state can be pushed from a single FM tower to cover a single metropolitan area. What towers should be used in conjunction to cover as many people as possible within a country? One possible direction is to exploit the fact that there appear to be regional CAs. For instance, some European countries provide certificates to their citizens: the inherently geographic scoping that FM transmitters provide would thus seem to be a natural fit for covering the vast majority of people who would be interested in learning of their fellow citizens' revocations.

Acknowledgments

We thank Adam O'Neill for his valuable feedback regarding our application of multi-signatures, and the anonymous reviewers for their insightful comments. We also thank Prabal Dutta and Thomas Schmid for their advice during the early stages of this project. This work was supported in part by NSF grants CNS-0643443, CNS-0917098, IIS-0964541, and CNS-1150177.

8. REFERENCES

[1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *Proc. International Cryptology Conference (CRYPTO)*, 1998.

[2] AlertFM. http://www.alertfm.com.

[3] M. Almeshekah. Proposal for better revocation model of SSL certificates. *Mozilla Foundation*, 2013.

[4] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Helman-Group signature scheme. In *Proc. Public Key Cryptography (PKC)*, 2003.

[5] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Proc. Asiacrypt*, 2001.

[6] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. IETF RFC-5280, May 2008.

[7] J. Douceur. The Sybil Attack. In *Proc. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[8] F. F. Elwailly, C. Gentry, and Z. Ramzan. QuasiModo: Efficient certificate validation and revocation. In *Proc. Public Key Cryptography (PKC)*, 2004.

[9] P. Evans. Heartbleed bug: RCMP asked Revenue Canada to delay news of SIN thefts, 2014. http://www.cbc.ca/news/business/heartbleed-bug-rcmp-asked-revenue-canada-to-delay-news-of-sin-thefts-1.2609192.

[10] I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and fresh certification. In *Proc. Public Key Cryptography (PKC)*, 2000.

[11] Google. CRLSets. http://dev.chromium.org/Home/chromium-security/crlsets.

[12] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 1983.

[13] P. C. Kocher. On certificate revocation and validation. In *Proc. Financial Cryptography (FC)*, 1998.

[14] A. Langley. No, don't enable revocation checking. https://www.imperialviolet.org/2014/04/19/revchecking.html.

[15] A. Langley. Revocation checking and Chrome's CRL. https://www.imperialviolet.org/2012/02/05/crlsets.html.

[16] P. D. McDaniel and A. D. Rubin. A response to "can we eliminate certificate revocation lists?". *Proc. Financial Cryptography (FC)*, 2000.

[17] S. Micali. NOVOMODO: Scalable certificate validation and simplified PKI management. In *Proc. PKI Research Workshop*, 2002.

[18] M. Myers. Revocation: Options and challenges. In *Proc. Financial Cryptography (FC)*, 1998.

[19] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. USENIX Security*, 1998.

[20] G. Neven. Efficient sequential aggregate signed data. In *Proc. Eurocrypt*, 2008.

[21] Y. Pettersen. The transport layer security (TLS) multiple certificate status request extension. IETF RFC-6961, June 2013.

[22] A. Rahmati, L. Zhong, V. Vasudevan, J. Wickramasuriya, and D. Stewart. Enabling pervasive mobile applications with the FM radio broadcast data system. In *Proc. International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2010.

[23] RDS Forum. *RDS Technical Specification*, 2009.

[24] T. Ristenpart and S. Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *Proc. Eurocrypt*, 2007.

[25] R. L. Rivest. Can we eliminate certificate revocation lists? *Proc. Financial Cryptography (FC)*, 1998.

[26] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, June 2013. IETF RFC-6960.

[27] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *Proc. ACM SIGOPS European Workshop*, 2004.

[28] N. Sullivan. The Heartbleed Aftermath: all CloudFlare certificates revoked and reissued, 2014. http://blog.cloudflare.com/the-heartbleed-aftermath-all-cloudflare-certificates-revoked-and-reissued.

[29] E. Topalovic, B. Saeta, L.-S. Huang, C. Jackson, and D. Boneh. Toward short-lived certificates. In *Proc. Web 2.0 Security & Privacy (W2SP)*, 2012.

[30] L. Zhang, D. Choffnes, T. Dumitras, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. In *Proc. ACM Internet Measurement Conference (IMC)*, 2014.

[31] L. Zhou, F. B. Schneider, and R. V. Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems (TOCS)*, 2002.