

Anonymous Datagrams over DNS Records

Samuel Brack Robert Muth Stefan Dietzel Björn Scheuermann

Humboldt-Universität zu Berlin, Germany

samuel.brack@informatik.hu-berlin.de, {robert.muth,stefan.dietzel}@hu-berlin.de, scheuermann@informatik.hu-berlin.de

Abstract—Many privacy-enhancing architectures require an anonymous communication channel as a component to acquire tokens or other material from a centralized authority without revealing the requester’s identity. While anonymous routing overlays like Tor can provide the required anonymity for TCP-like data streams, it increases dependence on third party infrastructure, and widespread use may overload the Tor network’s current capacity. Therefore, we propose to use requests in the Domain Name System as an anonymization layer by forwarding packets over public DNS resolvers operating in recursive mode. In contrast to existing approaches, we focus on UDP datagram transport. As an example use case, we implemented a system for anonymous issuance of blind signatures. We evaluate our domain-tailored solution against existing approaches like Tor and analyze its performance and privacy guarantees. Evaluation results show that the achievable data rate using the DNS-based anonymous channel is sufficient for many practical use cases, and that the provided anonymity is comparable to that provided by the Tor network.

I. INTRODUCTION

In the past, systems like Tor have been developed to provide secure anonymous communication for Internet users. In fact, deferring to Tor has become the default choice in many privacy-enhancing technologies (PETs) when an anonymous communication channel is required. Tor achieves privacy by implementing an overlay network where intermediate middle nodes forward requests to effectively anonymize the sender. As a result, the requested resource only sees the Tor network’s exit node as a communication partner. However, by using such an overlay network, users rely on the operation of these middle and exit nodes in the Tor network. Also, latency increases when using Tor, whereas bandwidth decreases due to Tor’s principle of *onion routing*. While Tor is designed to provide anonymous TCP connections for arbitrary use cases, it may not always be required to achieve anonymous communication.

By exploiting domain-specific knowledge about our expected traffic patterns, we implement a covert, anonymous channel using standard Internet protocols, reducing reliance on additional services, such as Tor. Typically, privacy-preserving systems rely on an anonymous channel for authentication, proof of knowledge of a token, or generally transfer of messages. These messages often comprise a single packet and follow a clear request-response pattern. Two or more requests should be unlinkable, and the receiver should not learn any information from their meta-data. In particular, the sender’s IP address(es) should not be revealed. For these traffic patterns, which cover a large number of practical use cases, we propose to employ the existing infrastructure of the Domain Name

System (DNS) for establishing anonymous message exchange. We did not design new Internet protocols but rather re-use old, established ones, diverting from their intended use cases. In doing so, we enable backwards compatibility and instant adoption in most real-world networks.

In this paper, we implement an anonymous channel for a privacy-preserving bonus point system for merchants [1] as a representative use case for our proposal. Based on the system’s instantiation for this specific use case, we discuss how it can be adopted to a range of similar use cases. An important step in issuing anonymous bonus points is the transfer of a secret token from a customer device to a central server called the *issuer*. The token holds information about a customer’s last purchase’s location, so it is desirable to hide any connection to personal metadata like the IP address of the token’s sender. To achieve anonymity, we encapsulate any communication between the customer and the issuer in DNS queries, in order to send these via arbitrary, third-party-operated DNS servers. The central server communicates with the third party in between and only sees the DNS server’s IP address. After the DNS packet reaches the issuer’s servers, they can also send a reply to the customer, which will be forwarded back to the customer’s smartphone. By that, the request-reply pattern is completed and the customer receives their bonus point without leaking metadata to the issuer. Other privacy-preserving systems that employ similar traffic patterns and require anonymous channels can be found in different areas like VANETs [2] or electronic voting systems [3].

Our main contribution in this paper is an anonymization system for datagrams over the Internet. We show that for bursty traffic with symmetric payloads, our DNS-based system outperforms Tor. The main idea is to rely on recursive resolution of TXT records by randomly selected public DNS servers, which resolve domain data controlled by the intended receiver.

By parallelization of requests and clever use of DNS protocol extensions, we accomplish a reliable and fast way to communicate without leaking metadata to the recipient of a packet. Our experiments show that DNS-based anonymization is efficient and preserves user privacy.

The remainder of this paper is structured as follows. In Section II, we identify and consider related work. After that, the use case and attacker model are presented in Section III. In Section IV, we describe the architecture and implementation of an anonymous channel over DNS. Following, in Section V an evaluation is presented, covering security, privacy, and

performance metrics. A comparison with Tor concerning our example use case has also been conducted. After that, we discuss our findings in Section VI before concluding the paper in Section VII.

II. RELATED WORK

A. Traffic anonymization

Anonymizing services have gained in importance in recent times, so there are many existing solutions for hiding IP addresses. The first system that aimed at hiding the sender's identity in an overlay network is Freenet [4]. This mixing network provides a non-realtime communication suitable for use cases like E-Mail or file transfer. A similar approach can be found in GUnet [5] where content is stored in a distributed hash table and addressed using a special URI format. However, in these systems, applications like web browsing or instant messaging are infeasible due to the resulting high delays.

Another approach in the evolution of anonymizing users on the Internet is the principle of onion routing. With onion routing, a path is used where routers only know their immediate neighbors but not the actual source or destination of a packet they process. This technique combines the responsiveness of an interactive web protocol with the anonymity even against Internet service providers on the way. The disadvantage of systems like Tor, which uses onion routing for anonymization, is an extensive and costly setup phase for a connection, or *circuit* in Tor's terms. While these costs are diminished if a circuit is used for a longer time (e.g., an entire web surfing session), a user wishing to send single packets with no relation suffers from this additional circuit establishment phase.

Garlic routing, a refined adoption of the onion routing idea, is implemented in the I2P anonymization network [6]. The essential idea of the garlic routing extension is the introduction of randomly distributed paddings per cell. One of Tor's main vulnerabilities is de-anonymization by an attacker with monitoring access to the network link from the onion proxy to the entry router and from the exit router to the target [7]. With data collected on these links, de-anonymization becomes more likely comparing patterns of bursts that are likely to be part of the same connection. By varying the sizes of packets randomly per link, combining frames in an apparent burst to one logical connection becomes more unreliable.

Over the years there have been numerous other approaches to create anonymous communication on the Internet. An interesting approach is the usage of covert channels, where traffic is hidden in noise or protocol messages that are sent anyways. [8] provides an overview over such covert channels. Some of the approaches discussed there can also be used to provide a channel for anonymous communication, for example approaches based on broadcasts (and possibly fake source addresses) or on redirects in application layer protocols. However, some of these approaches rely on a local network in case of broadcast reachability or other details in their setup. Even online games can be used to anonymously transfer information [9] albeit the achievable data rates are very low.

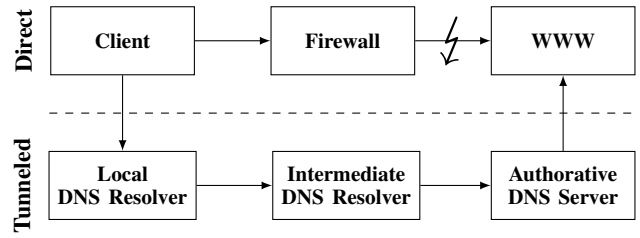


Fig. 1: DNS tunneling in a restricted network

B. DNS tunneling

The Domain Name System was originally designed to resolve human-readable hostnames to IP addresses. Over the years, there have been many additions to that protocol, for example for supporting new protocols like IPv6 and security features like DNSSEC [10]. Another addition to DNS is the support for arbitrary data which can be stored in TXT records. So, by resolving a domain name the DNS server can reply an printable ASCII text instead of an IP address.

A rather well known creative (ab)use of DNS is the use of DNS tunnels (see Figure 1) [11], [12]. DNS tunnels are designed to provide free Internet access for users that are connected to a pay-per-use WiFi network by bypassing common filter techniques. Generally, such networks allow anyone to connect to an access point (i.e., access is not restricted by a WEP/WPA passphrase), but they redirect every HTTP request to a landing page, called a *captive portal*. The user is then requested to pay for Internet access. HTTP requests to the World Wide Web (WWW) are intercepted by a firewall, so no matter what the user requests in their web browser, they will only gain access to the captive portal until user authorization is completed. For providing the captive portal itself, which is a standard web page, the firewall usually allows DNS requests. Replacing DNS traffic would result in problems on the end user's device even after successful authorization. Many devices are configured to cache DNS replies. A captive portal that replaces DNS traffic thus could pollute the customer's DNS cache leading to unexpected problems after authorization at the portal.

So, these DNS requests can be used to reach the Internet in general and, more specifically, a personal server for tunneling. Both the sender and the receiver have to encapsulate their actual traffic in DNS requests and responses. All requests from behind the firewall will be forwarded via the DNS tunneling to a certain authoritative DNS resolver outside the firewalls network. It's not unusual to use the local and intermediate DNS resolver to reach the authoritative DNS resolver, in order to fulfill the firewall's rules. Thenceforth the actual traffic can be sent to the actual receivers by encapsulating it in DNS packets. DNS replies will be encapsulated again and sent back to the requester behind the firewall, using the same pathway backwards.

However, to our knowledge there is no implementation of DNS tunnels used to mask a host's IP address with respect

to the DNS server. DNS tunnels do not rely on recursive record resolution. Thus, they cannot guarantee anonymity for the client, because records are resolved by the client and not just forwarded. In fact, iodine and other DNS tunnel software often requires users of the tunnel to authenticate themselves at the tunnel endpoint.

III. SYSTEM MODEL

A. Example use case: anonymous bonus points

As an example use case for our proposed protocol, we consider anonymous bonus point systems, which are representative for other protocols with similar communication requirements. Without proper anonymization, user profiles in bonus point systems may expose personal data like buying behavior or personal interests. A number of proposals exist that introduce fully anonymous bonus point programs [13], [14]. While Blanco-Justicia *et al.* consider using an existing anonymous channel just like Tor for their implementation of a bonus point program, Enzmann *et al.* do not address this issue.

In an earlier paper, we presented an anonymous bonus point system [1] based on David Chaum’s proposal for electronic cash using blind signatures [15]. In the following, we summarize how a new bonus point is issued to understand the requirements for an anonymous communication channel. At the beginning, a merchant receives a certain amount of unique *vouchers* from a central trusted party, as shown in step 1 in Figure 2. In our scenario, the trusted party is called *issuer*. After a customer completes a purchase of goods or services at a merchant’s checkout, they receive some of these *vouchers* on their smartphone in step 2, depending on how much money they spent. These *vouchers* are used to “pay” for the generation of anonymous bonus points. The *voucher* handover occurs unidirectional, e.g., via QR code or NFC, so the merchant doesn’t get any personal information about the customer. Bonus points are digital tokens that are generated locally on the customer’s device. Essentially, they consist of a randomly selected serial number with enough entropy to avoid collisions with other users. In the next step, the customer obtains a *blind signature* from the issuer in exchange of a *voucher*. For that, the customer *blinds* the bonus point serial number. Blinding is an operation in asymmetric cryptosystems, and the exact implementation depends on the cryptosystem used. This operation is non-reversible for anyone not knowing

the secret blinding factor. Widely known systems that support blinding are RSA and ElGamal [15].

Next, this blinded number and the *voucher* are sent to the issuer using an anonymous channel (step 3 and 4). The issuer provides a digital signature for the blinded bonus point. The issuer checks whether the *voucher* has been already redeemed. If not, the signature is sent back to the customer who can unblind it after steps 5 and 6. By unblinding the issuer’s signature becomes valid for the unblinded bonus point serial number.

It is essential that the issuer learns no metadata about the customer in steps 3–6, as the customer reveals at which merchant he purchased goods. One metadata that is leaked by an Internet communication is the customer’s IP address. Thus, anonymization requires a communication channel that is either not using the customer’s real IP address or using no IP address at all.

B. Attacker model

In our system, we aim to protect against the end point of our communication path. For our use case, this end point is represented by the issuer of the blind signatures. This end point shall not learn any identifying information about the sender of the data packet. It has full control over its network stack and can retrieve all information provided by any protocol involved in the communication. Additionally, the issuer is able to store all received messages indefinitely and tries to match messages to user identities in order to create “shadow profiles” of its users.

The content on the transport link from the customer to the issuer is considered to be openly readable to any honest-but-curious middle box on the network path. However, it is assumed that at least one of these intermediate links is controlled by someone unrelated to the issuer. Of course, the intermediary node learns both IP addresses, and it can even intercept the whole communication. Nevertheless, the messages between the client and the server do not contain any personal or valuable information for the interceptor, as a result of the blind signatures. Also the middle boxes do not have any reason for manipulating or withholding requests or answers, because they would not gain any additional benefit. The customers can re-submit the requests when they determine missing answers for their earlier requests or any manipulations by checking cryptographic consistency.

IV. ANONYMOUS DATAGRAM ARCHITECTURE

After understanding the requirements for an anonymous datagram channel, the construction of a DNS-based anonymization channel is described next. There are three relevant parties involved as displayed in Figure 3. All IP addresses are chosen as example values. The addresses in boxes are the host’s addresses, the addresses at the arrows represent the addresses in the IP packets sent between the hosts. In addition to the sender’s client and the receiver’s server there exists a DNS resolver in the middle. This resolver can be any public DNS resolver, there are no special requirements

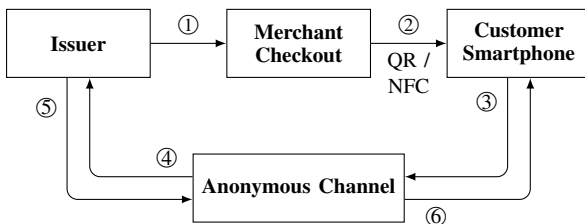


Fig. 2: Coin signing scenario

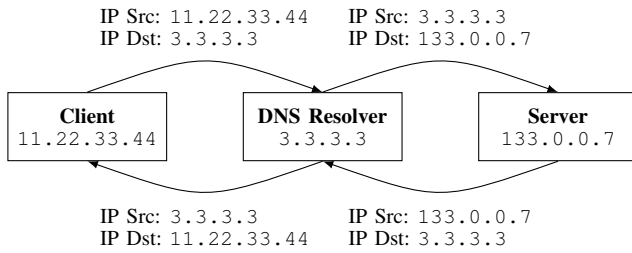


Fig. 3: Hiding the client IP address by forwarding traffic via a DNS resolver

to be considered for anonymization. The advantage of using public DNS resolvers is that users can select them freely and do not depend on volunteers to operate specialized servers. Its purpose is to relay traffic between sender and receiver in order to mask the sender's IP address. In our scenario, the sender wants to transmit a request of limited length to the receiver where the request is answered with an equally short and unique reply message. After the reply message has been received by the sender's client, the entire communication channel can be discarded.

The DNS resolver in the middle performs the role of a router by forwarding all traffic between the two communicating parties. This routing happens on the application layer, which means that the actual IP addresses of both communicating parties are not present in the IP header received by the other party. Because the resolver cannot answer the request itself, it will be forwarded to our actual recipient, the authoritative DNS server. By forwarding the still unanswered request to the authoritative DNS server, the resolver uses a new IP connection, hence using its own IP address as source address. So, the authoritative DNS server is not communicating with the client's IP address. Answers to the DNS requests results will be forwarded by the DNS resolver in the middle. The IP packet headers are also displayed in the figure. Because DNS was not originally intended to serve as a router, some protocol options have to be set to ensure the DNS resolver acts as intended. These options are set by the client's computer who initiates the entire communication process.

The most important modification is to ensure recursive operation of the DNS resolver. Under normal circumstances, DNS resolvers operate in iterative mode by pointing the requesting party to the authoritative DNS server for the next subdomain. That server has to be queried by the requester itself until the entire domain is resolved. In contrast, recursive mode means that the DNS server itself resolves the entire domain and returns the result of its (possibly costly and time-consuming) querying to the requester. Typically, public DNS servers do not resolve requests recursively, even when demanded in the request flags. This is due to the additional overhead for the resolver, because it potentially has to ask more than one following DNS resolver to get the final answer. During our study, we noticed that DNS queries containing TXT requests, however, mostly have been answered recursively (as

specified in the requests), even from DNS resolvers which normally do not resolve recursively for IP addresses.

Based on this observation, we have built UDP datagrams with DNS queries containing our actual payload in the "query domain" field as follows:

```
LINK(Payload=
  IP(Src=11.22.33.44, Dst=3.3.3.3, Payload=
    UDP(Src_Port=1122, Dst_Port=53, Payload=
      DNS(ID=123, Flags=[Lookup: Recursive,
        ECS_EDNS_Subnet=0.0.0.0/0], Queries=[(Type:
          TXT, "QUERY_PAYLOAD.dns.issuer.tld")]))))
```

This query is sent to the DNS resolver 3.3.3.3, which forwards the request to the authoritative DNS server of `dns.issuer.tld`, which is our final receiver. The DNS header field "ID" sets a user-selectable message ID to match incoming answers to previous requests. Setting the flag "Recursive" requests recursive lookup by the DNS server. By requesting a subdomain `QUERY_PAYLOAD`, we can send arbitrary content to the authoritative server, which is then answered in the reply message. Because the payload is also subject to the general domain restrictions, it is limited to a maximum of 251 ASCII characters, less the domain length. Note that the EDNS subnet specification is not a real DNS flag but an additional resource record. This resource record is required for keeping the anonymity of the sender between the DNS resolver and the authoritative DNS server. The authoritative DNS server can specify the origin subnet of a request for geographical distinction of DNS queries [16]. By setting the subnet to `0.0.0.0/0` the client can suppress leakage of that information. The corresponding DNS reply for the original sender could look like:

```
LINK(Payload=
  IP(Src=3.3.3.3, Dst=11.22.33.44, Payload=
    UDP(Src_Port=53, Dst_Port=1122, Payload=
      DNS(ID=123, Flags=[Answers,
        ECS_EDNS_Subnet=0.0.0.0/0], Queries=[(Type:
          TXT, "QUERY_PAYLOAD.dns.issuer.tld")],
        Answers=[(Type: TXT, "ANSWER_PAYLOAD")]))))
```

The headers in the reply message replicate the request message. The answer to the actual request is provided as a resource record located at the previously requested subdomain. It is also limited to strings and the overall maximum length is given by the maximum size of a UDP datagram. In order to quickly generate packets according to that specification it is possible to use Dig with these parameters:

```
dig -t TXT +subnet=0.0.0.0/0 +recurse
@3.3.3.3 QUERY_PAYLOAD.dns.issuer.tld
```

This is also the exemplary parameterization for our generated requests in the following performance measurements. We sent unique query payloads via various DNS resolvers in different time spans.

V. EVALUATION

After implementing a prototypical server infrastructure of the issuer, we evaluated our concept in several dimensions. For that, a high-performance client simulator that requests random subdomains has been implemented in Python. An authoritative DNS server located in our university network (with sufficient connectivity to the Internet) was installed. That server receives DNS requests for arbitrary subdomains of its administered domain and performs a hashing operation of the subdomain string to generate payload for the reply. After that, the server returns the response to the requesting IP address. By using a cryptographic hashing operation a pseudo-random reply is enforced, so that DNS caches cannot predict or cache any data in our experiment. This ensures a measurement of the true performance of our proposed system.

A. Performance measurements

In order to gain a rough overview over the achievable performance of our proposal we conducted tests to determine the throughput of our system. Although our use case does not require huge data rates, we can provide an overview of how long a typical consumer transaction will take based on our measurements.

We have set up an authoritative DNS server using Python 3.6.3 and dnslib 0.9.7. The server only answers to TXT record requests with the cryptographic hash value of the requested subdomain, so we can simulate similar traffic like blind signatures would. Additionally, DNS cache servers or other middleboxes cannot predict traffic patterns or distort the experiment in another way. The answers fill out the maximum of transferable DNS response payload and are padded to 200 bytes, for reaching higher data rates. Next, we generated 1000 unique DNS requests and sent them to our own authoritative DNS server using random public DNS resolvers located in Germany, from a publicly available directory [17]. The requests and responses were captured in a PCAP file, so we could replay the exact requests with different data rates. We can adjust the sending data rates by defining the sending interval of the UDP packets. The requests were sent from a computer located outside of the university facilities to arbitrary DNS resolvers, which are forwarding the requests to our DNS server at our university lab. The direct round-trip-time (RTT) between our authoritative DNS server and the end-user client is in the range between 30 ms and 40 ms.

Our results show the amount of requests we could send in several time spans, which are defining the resulting data rates. We checked each response for correctness and calculated the overall resulting data rate by adding all payload data together, divided by the amount of successfully transmitted requests and responses. Table I shows the mean values of 10 runs.

In the experiment, we utilized six different sending durations. Due to the influence of randomness in the network and the choice of used DNS resolvers the amount of sent requests and responses are different for each run. Also the locations and spreading of the DNS resolvers influences the measured results. The results in the table show the decreasing response

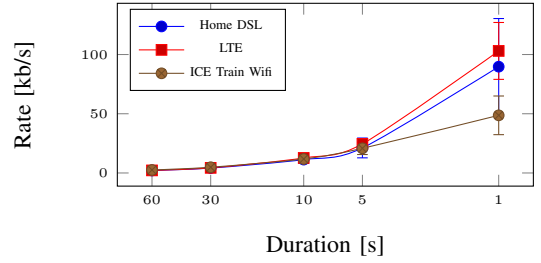


Fig. 4: Achievable data rates for 1000 DNS requests, $n=10$

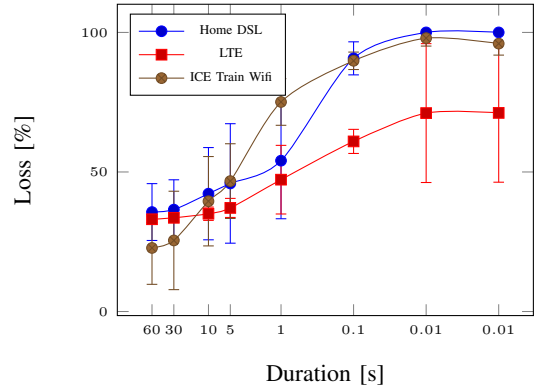


Fig. 5: Packet loss for 1000 DNS requests, $n=10$

ratio with faster sending rates. The shorter the sending duration becomes, the higher the sending rate will be, but the loss rate rises rapidly. We observed that DNS resolvers don't forward that much traffic indefinitely, so they might purposely ignore requests at some point. For example, the public Google DNS servers (located at 8.8.8.8) temporarily blocked our requests if more than about 15 fully filled TXT queries were sent in a burst. This throttling behavior is caused by configurable rate limits in DNS servers, like BIND accomplishes with token buckets [18].

As you can see in Figure 4, we can reach sufficient data rates for our use case. The error bars indicate the standard deviation of the data rate values. Even submitting e.g. 100 vouchers (one DNS request each) for blind signing at the issuer can be achieved in about one second with a data rate of 100 kbyte/s plus the RTTs. Faster transmission results in higher receiving loss rates. Figure 5 shows the aforementioned throttling and networking bottlenecks resulting in high packet loss. Sending

	60s	30s	10s	5s	1s	0.1s
Requests	658	650	634	626	568	512
Responses	644	635	578	541	460	93
Paket Loss Rate	36%	37%	42%	46%	54%	90%
Rate in kbyte/s	2	4	11	21	90	181

TABLE I: Mean values of 10 runs of the time performance evaluation with a 51,4 Mbit/s $\downarrow$ 10 Mbit/s $\uparrow$ standard Internet connection (DSL)

1000 packets in 0.1s results in loss rates over a half.

B. Security evaluation

Another important dimension in our evaluation is data security and privacy. The original concept of using IP-based transport protocols is sensible, as IP traffic can be generated by mobile devices almost anywhere on earth. With that constraint, a focus on masking the sender's IP address is set.

As specified in the attacker model, all involved parties have different objectives on how to gain unintended knowledge from the system. The communication's content itself, consisting of cryptographically blinded data and a voucher with very limited information, is considered to be not threatening to the customer's privacy. Only in conjunction with transport metadata (i. e., the sender IP address) a match between a single purchase and this IP address could be created. External parties – especially DNS resolvers – can indeed capture payload and IP addresses, but they lack knowledge of the specific appearance of the payload. The vouchers are pseudonymous themselves and a database located at the issuer's is necessary to uncover their origin. Thus, our proposal hides the IP address from the only party that is able to match these data records – the issuer.

The packet content itself can be additionally protected by encrypting it using standard symmetric or asymmetric encryption algorithms. Symmetric encryption prohibits the DNS resolvers to read the contents, unless they obtain the key. So, in our example use case it would be necessary to store the key on the clients. Because symmetric encryption uses the same key also for decryption, an attacker could extract the symmetric key from the client application and use it after intercepting any traffic. Asymmetric encryption allows to send encrypted datagrams without exchanging keys before actual communication. Only the receiver with the private key can decrypt them. So, even if a DNS server intercepts the traffic and obtains the public key from the client application, it cannot decrypt the message. But for returning an answer another key is needed, resulting in the same difficulties as with symmetric encryption.

Very recent attempts to secure DNS itself describe using HTTPS for securing DNS requests [19]. HTTPS is a long-established protocol for securing data transfers over the Internet, which implements perfect forward secrecy (PFS). The main idea of PFS is the foregoing key exchange before establishing an encrypted communication channel. It is designed to securely exchange dynamically generated encryption keys, using a so-called long-term secret. So, it would not be necessary to distribute constant encryption keys, or even implementing further encryption methods by ourself, for securing the packet contents. However, key exchanges require more than one additional datagram until the encrypted data can be transferred.

Despite using blind signatures for securing the integrity of voucher transfers, it is also possible to add cryptographic signatures for securing the datagram contents. Besides encryption, HTTPS also implements mechanisms for ensuring data integrity and authentication. Man-in-the-middle attacks –

which are a serious risk for key exchanges – can be prevented by integrity and authentications checks. For our example use case, we rely on the security given by blind signatures, so we do not have to send more than one datagrams for data exchange.

During our performance evaluation we also observed the use of EDNS0 between some of the DNS resolvers and our authoritative DNS server, which leaked information about the IP subnet the requester is located in. In our case the requester's ISP assigned an IPv4 /24 subnet. So by revealing the subnet's IP address space, the issuer could link the requests together and may guess the IP address by a chance of $\frac{1}{253}$ (255 addresses in a /24 network less the network and broadcast addresses). Additional constraints could be used to de-anonymize users even further, for example analysis of actually reachable IP addresses from the respective subnet by sending ICMP echo requests. As explained above, is it possible to prevent this behavior by specifically setting the EDNS-Client-Subnet (ECS) to 0.0.0.0/0. There is still no guarantee the DNS resolver will follow the standard and keeps the requesters IP address secret, but in our evaluation all DNS resolvers which used EDNS0 applied the ECS settings correctly. Usage of EDNS0 was only found in two instances in our list of public DNS resolvers. Therefore it is still recommended to swap DNS resolvers for related requests each time, so the risk of deanonymization can be reduced further.

C. Communication indirection

The first and most obvious effect of the DNS anonymization technique is the elimination of direct communication with the issuer. Every packet is relayed through one or more DNS servers that are controlled by organizations other than the issuer. Theoretically, the issuer could collaborate with a DNS server provider in order to gain information about both the purchase-specific voucher and the sender's IP address. In order to gain personal information by collaborating with a DNS server provider the issuer has to dictate (or at least influence) which DNS servers are used by the customer. However, the DNS server used for a transfer of a packet is selected only by the customer. An even lower attack surface can be achieved by selecting different DNS servers per packet, so that one compromised server only leaks information about that single packet's sender. Unlike in other anonymization networks, switching DNS servers comes at almost no overhead cost because only the receiver address of a UDP packet has to be rewritten.

D. Reliability

Because only recursive resolving yields proper anonymization and unlinkability, it is important to not resolve the TXT request iteratively with the customer's IP address. In the DNS header the customer can detect whether a resolver's answers are processed recursively or not. If not, the resolver's answer contains either an error notification or a redirection to the next DNS resolver. So, a customer can check if the selected DNS

resolver operates in recursive mode or not and can switch to another DNS resolver.

On the downside, this also means DNS resolvers could block the proposed approach very easily by either prohibiting all recursive requests no matter of which type or by blocking TXT requests in general. For effectively blocking traffic from our proposed system it would not be enough to just restrict TXT requests with a token bucket or other load balancing techniques because each burst of requests can be distributed over several different DNS resolvers. By that, single TXT requests should not be affected by individual limitations. To entirely block our traffic, DNS resolvers could blacklist our authoritative DNS domains, but these are easily replaceable.

We also verified whether there is no difference when using IPv6 instead of IPv4. We set up our own authoritative DNS server to also listen for DNS requests using IPv6 and used IPv6 supporting third party DNS servers for connectivity tests. As expected, there is no difference regarding the reliability or security of our proposal. The forwarding of the request and response by recursive lookup both work as desired.

E. Resources consumption

In order to check whether forwarding a lot of TXT requests at a DNS resolver over-saturates its capacities, we have conducted an experiment with realistic data for our use case. The largest German bonus point system Payback claims to generate about 1000 points per second (one point is awarded for one Euro of revenue) [20]. We sent 1000 requests to a single DNS resolver to stress-test it and evaluated the CPU load. The resolver is located at our university lab and is running Bind 9.10.3. We used 10 cloud servers that generated 100 TXT requests each, submitting them simultaneously via our experimental DNS resolver to our experimental signing server. It took 53 ms to receive all 1000 requests which were handled without excessive resource consumption. The maximum CPU load observed during the experiment was 50 percent for a period of 213 ms. We note that the virtual machine running the DNS server was left in a default configuration without any optimizations. We conclude that a properly-configured public DNS server would not be exploited significantly by our approach. Besides, our approach of unlinkability requires sending requests via different DNS resolvers, which distributes the load between several servers. It is also noteworthy that even opening a normal website can generate dozens of DNS requests, as scripts, advertisements, and embedded content is often located on different domains.

F. Comparison with Tor

As discussed in the related work section, another approach for solving the indirection problem would be the usage of the existing Tor network. However, on closer inspection of the exact mechanics of Tor, the performance of processing several unlinkable request-response-pairs in a short period of time becomes rather low.

In our proposed system each bonus point is generated independently in order to guarantee the unlinkability of several

vouchers to one purchase. By linking vouchers, the issuer could derive information about the amount of a purchase. When vouchers are used independently, there is no leakage of which vouchers are connected in one bill and which ones originate from separate purchases. A consequence of this unlinkability is that every voucher (and blinded signature request) has to be sent independently on the network, i.e., not in one logical “connection”. In case of Tor, this would imply the establishment of a new circuit for each voucher so that a new exit node can be selected. Otherwise, the issuer could easily link vouchers by keeping track of their TCP connections on which they arrive and ascribing those with the same connection.

However, if unlinkability is guaranteed by other means (or not important for a specific use case), Tor’s security guarantees are very high. In contrast to our system, Tor protects against a malicious node in any circuit, because a node does not learn anything about either content or metadata of a circuit. Only if several nodes cooperate, they can perform attacks that uncover some metadata, but these attacks are expensive and require positioning in all areas of the Internet [21].

Setting up a new circuit in Tor is a rather time consuming process. First, a directory server is queried for a list of available onion routers. After that, a connection is established to the entry node by opening a TCP connection with a three-way handshake and a `Circuit create` message. The connection to the second and third onion router are created by issuing a `Circuit extend` message to the last onion router in the existing chain. Each new step results in a three-way handshake and a message for creating the connection. Only after the connection to the exit node has been set up a query can be sent towards the actual destination server. Additionally, for every step a message is passed to the user’s onion proxy so that a symmetric key can be exchanged. So, for establishing a complete Tor connection with an entry, a middle and an exit node at least 12 roundtrip times are needed. Any delay for querying the directory servers is not yet included in this calculation, as the directory can be cached at least for some time on the user’s computer.

For evaluating the performance of our datagram-oriented use case compared to our proposed communication channel, we also ran our performance evaluation with Tor. First, we modified the Tor client source code, so swapping circuits is not limited by the `MAX_SIGNEWNYM_RATE` rate limit (default 10 seconds between every new circuit). Following, we submitted the exact same request payloads via Tor as in the foregoing performance evaluation and swapped the circuits after each request. We did not use DNS for that, instead we used HTTP

	Ø Rate	Standard Deviation	Retransmissions
Sending	355 byte/s	$\sigma = 40$	0.06%
Receiving	298 byte/s	$\sigma = 34$	0.36%

TABLE II: Mean values of 10 runs of the time Tor performance comparison, requiring on average 682s ($\sigma = 103$)

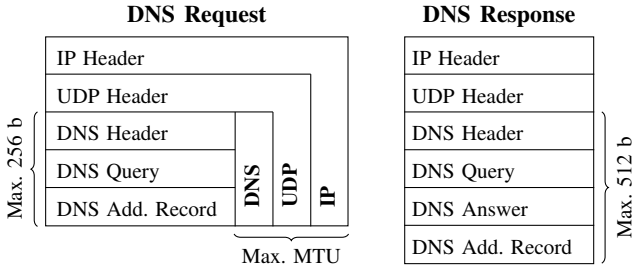


Fig. 6: Protocol stack of DNS in UDP datagrams

via Privoxy, Socks and our modified version of Tor client 0.3.4.0-alpha-dev. Even though Tor provides an interface for querying DNS, we were not able to query TXT records with it. As illustrated in Table II, using a standard Internet connection we could submit 1000 requests at an average of 11 minutes 37 seconds resulting in an overall receiving data rate of 298 byte/s. Due to the design of TCP used by Tor no packet got lost permanently, but retransmitted. An analysis of the endpoints shows that an average of 338 different Tor exit nodes reached our server ($\sigma = 7$, min. 323, max. 347). The statistics service “Tor Metrics” showed a number of approximately 800 active exit nodes during our measurements [22]. Although many exit nodes were apparently re-selected, the inner structure of the constructed circuits differed for each request. Theoretically, we could reach higher data rates by parallelizing multiple Tor instances on a client with different circuits, but doing so would over-saturate the Tor network disproportionately in our opinion. The fact that we had to modify the Tor client to circumvent the circuit creation limit also shows that this approach is not encouraged by the Tor community. In conclusion, it is evident that using Tor for sending small packets that are unlinkable for the receiver is costly and resource-consuming.

VI. DISCUSSION

A. Limitations

The DNS protocol uses UDP datagrams for transmission, but the RFC standard also allows TCP connections for requests which answers are too long for UDP datagrams. Nevertheless, the standard doesn’t explicitly enforce TCP support for DNS resolvers [23]. As illustrated in Figure 6, DNS requests and responses are encapsulated by Internet Protocol (IP) packets whose sizes are limited to the variable Maximum Transmission Unit (MTU). This limit is not affected differently whether IPv4 or IPv6 are used. DNS requests are limited to a maximum domain name length of 253 characters, while each segment of it may only have 63 characters. The maximum transmittable data would fit in four groups of standard letters: (63) . (63) . (63) . (62) . This restriction even applies with the Extended DNS (EDNS0) option, which permits for an increased maximum UDP datagram size up to 512 bytes [24]. Therefore, a DNS request query can contain 251 bytes of ASCII data, less the domain length of the authoritative DNS server. The DNS reply can be free text, so a regular UDP

datagram with its own header (8 bytes), the IP header (20 bytes) and DNS header (12 bytes) can contain up to 484 bytes for queries and answers. In case a request reaches its maximum size, the corresponding reply would contain the requests query itself and up to 200 bytes for the answer (plus 13 bytes for additional DNS headers). It is unthrifty for bidirectional communication to repeat the query in the response and for our use case of no avail. Repeating the query is important for the intended use of DNS, so the receiver of a DNS response can correlate the answers to the queries supposing the request contained more than one. Unfortunately, we cannot suppress this data reflection behavior when querying the DNS resolvers.

B. Generalization

In the previous sections we showed how to implement an anonymous channel for UDP traffic in our specific use case. As mentioned above, this concept can be generalized and applied to any other type of problem where some kind of anonymous token is passed over the Internet. The length of one request and reply should not exceed the limits of a UDP packet. Otherwise, an ordered reliability system would be necessary in order to guarantee a seamless flow of data and to deal with packet loss.

As long as the end user is able to freely choose (and communicate with) their DNS resolver our system works. Many real-world firewalls do not impede DNS traffic because end-user DNS caches would potentially suffer from these restrictions. Therefore, our anonymization approach works in almost all scenarios where some kind of Internet connectivity is provided. Because one of the most important application-layer protocols is used, restrictions like carrier-grade NAT or port blocking are designed to not interfere with this kind of traffic.

VII. CONCLUSION AND FUTURE WORK

We propose an anonymous communication channel optimal for blindly signing short messages or transmitting authentication tokens, in our case digital bonus points. Optimal in this context means low latency and minimal effort for establishing the communication channel. Anonymity is accomplished by using third parties in between of the sender and receiver for hiding IP addresses and through cryptographic algorithms for securing the contents. Users can communicate directly to a final destination – without hiding their IP address but still using blind signatures – or via an unrelated and arbitrary third party. The results in our evaluation show the capability of sending and receiving several hundred messages in very short times, due to the absence of complex connection establishment protocols. Unlinkability is also achieved by sending messages via many different routes by using various third party DNS servers in between. We have shown to overcome relatively high loss rates of our packets by detecting loss quickly and retransmitting via another DNS resolver in case of a loss event. Our proposed system uses the DNS system, so it is simple to adapt and does not require any new protocol designs. We have also shown the performance advantage compared to Tor. Outsourcing the anonymity setup by using Tor is a very

good idea in context of security, but our use case requires unlinkability between a lot of single datagrams and therefore a lot of different circuits. Because establishing these circuits is the most time consuming part of Tor, use cases like ours would require very long run-times to create a new circuit for every request.

After developing the basic idea of anonymous DNS channels, several ideas to extend this proposal come to mind. For example, it might be interesting to cooperate with a public DNS server operator in order to gain more insights on performance and operational implications of our idea. By bringing attention of this anonymization tool to these operators they would also be able to finetune their intrusion detection systems so that they do not classify such traffic as an attack on their infrastructure.

ACKNOWLEDGEMENT

We would like to express our gratitude to the BMBF (German Federal Ministry of Education and Research), who funded this research in the context of the *Goodcoin* project (Project Number 16KIS0376).

REFERENCES

- [1] S. Brack, S. Dietzel, and B. Scheuermann, "ANONUS: Anonymous bonus point system with fraud detection," in *LCN*, IEEE Computer Society, 2017, pp. 356–364.
- [2] F. Schaub, F. Kargl, Z. Ma, and M. Weber, "V-tokens for conditional pseudonymity in VANETs," in *WCNC*, IEEE, 2010, pp. 1–6.
- [3] M. Burmester and E. Magkos, "Towards secure and practical e-elections in the new era," in *Secure Electronic Voting*, ser. Advances in Information Security, vol. 7, Springer, 2003, pp. 63–76.
- [4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Workshop on Design Issues in Anonymity and Unobservability*, ser. Lecture Notes in Computer Science, vol. 2009, Springer, 2000, pp. 46–66.
- [5] *Gninet framework for secure peer-to-peer networking*. [Online]. Available: <https://gninet.org>.
- [6] *I2P Anonymous network*. [Online]. Available: <https://geti2p.net/en/>.
- [7] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *IEEE Symposium on Security and Privacy*, IEEE Computer Society, 2005, pp. 183–195.
- [8] S. Zander, G. J. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 1-4, pp. 44–57, 2007.
- [9] S. J. Murdoch and P. Zielinski, "Covert channels for collusion in online computer games," in *Information Hiding*, ser. Lecture Notes in Computer Science, vol. 3200, Springer, 2004, pp. 355–370.
- [10] S. Weiler and D. Blacka, "Clarifications and implementation notes for DNS security (DNSSEC)," RFC Editor, RFC 6840, Feb. 2013. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc6840.txt>.
- [11] L. Nussbaum, P. Neyron, and O. Richard, "On robust covert channels inside DNS," in *SEC*, ser. IFIP Advances in Information and Communication Technology, vol. 297, Springer, 2009, pp. 51–62.
- [12] *Iodine DNS tunnel*. [Online]. Available: <https://code.kryo.se/iodine/>.
- [13] A. Blanco-Justicia and J. Domingo-Ferrer, "Privacy-preserving loyalty programs," in *DPM/SETOP/QASA*, ser. Lecture Notes in Computer Science, vol. 8872, Springer, 2014, pp. 133–146.
- [14] M. Enzmann and M. Schneider, "A privacy-friendly loyalty system for electronic marketplaces," in *EEE*, IEEE Computer Society, 2004, pp. 385–393.
- [15] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, Plenum Press, New York, 1982, pp. 199–203.
- [16] C. Contavalli, W. van der Gaast, D. C. Lawrence, and W. Kumari, *Client Subnet in DNS Queries*, RFC 7871, May 2016. [Online]. Available: <https://rfc-editor.org/rfc/rfc7871.txt>.
- [17] *Public DNS server list*. [Online]. Available: <https://public-dns.info/>.
- [18] *Using the response rate limiting feature in BIND 9.10*. [Online]. Available: <https://kb.isc.org/article/AA-00994/0/Using-the-Response-Rate-Limiting-Feature-in-BIND-9.10.html>.
- [19] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," Internet Engineering Task Force, Internet-Draft, Jun. 2018, Work in Progress, 21 pp. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-ietf-doh-dns-over-https-12>.
- [20] *Payback Facts & Figures*. [Online]. Available: <https://www.payback.net/en/about-payback/facts-figures/>.
- [21] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: routing attacks on privacy in tor," in *USENIX Security Symposium*, USENIX Association, 2015, pp. 271–286.
- [22] The Tor Project, Inc. (2018). Tor metrics, [Online]. Available: <https://metrics.torproject.org>.
- [23] R. Braden, "Requirements for Internet hosts - application and support," RFC Editor, STD 3, Oct. 1989.
- [24] P. Vixie, "Extension mechanisms for DNS (EDNS0)," RFC Editor, RFC 2671, Aug. 1999. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2671.txt>.