DNS++: Dynamic Name Resolution with Homomorphic Encryption Based Privacy

Francesco Tusa*
School of Computer Science and Engineering
University of Westminster
London, United Kingdom
Email: f.tusa@westminster.ac.uk

David Griffin and Miguel Rio
**Department of Electronic and Electrical Engineering*
University College London
London, United Kingdom
Email: {francesco.tusa, d.griffin, miguel.rio}@ucl.ac.uk

***Abstract*—This paper presents DNS++, a re-design of the Internet's name resolution system that addresses dynamic information and privacy. DNS++ uses a pub/sub overlay to send updates about a given service to interested clients, allowing them to (re)select between replicas according to their requirements, as updates about services and their features dynamically change. Since third-party brokers in the overlay are not always trusted for the confidentiality of the content flowing through them, clients' privacy is preserved in DNS++ through homomorphic encryption. Brokers are prevented from accessing encrypted service information but can perform homomorphic match and forward service updates to relevant clients through the overlay accordingly. Assuming that forwarding tables in each broker are implemented via ordered data structures, the time required for adding a new client's subscription, and to perform homomorphic match between existing subscriptions and service updates, would grow logarithmically with the number of entries within a table. This is shown by our performance evaluation, which confirms that DNS++ is feasible to be deployed with an acceptable performance overhead.**

***Index Terms*—DNS, homomorphic encryption, privacy, pub/sub.**

I. Motivation

The Domain Name System (DNS) [1] has been an integral part of the Internet architecture since the early 1980s. Although, in theory, two end points can establish a communication without help of a name resolution, this hardly ever happens. DNS was a crucial sub-system to allow the Internet to scale and to become more user friendly; however, despite its age, it has hardly changed in the last 40 years, apart from some extensions and the advent of content distribution networks. Fundamentally, DNS has two major problems: static responses and lack of privacy. Once a query is resolved, there is no way for a client to receive updates for a given name. This makes resilience, load-balancing and mobility difficult to implement. One would like to have new replicas, or new metadata about those replicas, to be continuously sent to interested clients; these would then have the freedom to improve their quality of experience accordingly. Moreover, the queries are processed by DNS servers that can easily tie a given user to a particular name resolution. Albeit this is a major privacy concern identified by the community, there are only limited available solutions today.

Privacy is important in the Internet of Things (IoT). There are two main roles for dynamic name resolution in IoT. Firstly, for gateways to resolve the names of service and application processing functions to the locators of specific service replicas. To increase resilience and aid localisation many services will be replicated. A new mechanism is needed that allows gateways to subscribe to service names and subsequently to make dynamic decisions on which instance is most suited based on updates generated as replicas are deployed and come on- and off-line. The second role is in applications and users resolving the names of gateways supporting specific sensors and actuators. In both roles, proper security mechanisms are needed to ensure that the resolution infrastructure is unable to identify the names or identities of the IoT services or gateways while subscriptions and publications are matched.

While DNSSEC [2] can only prevent attackers from manipulating or poisoning the responses to DNS requests, DNS over HTTP [3] and the work in [4] are two proposals from the community to implement privacy for DNS queries; however, they require client queries to be processed by third parties, who will thus be able to access and view the related content. Although the solution in [5] solves this problem using a Private Information Retrieval (PIR) scheme, it does not support the dynamic distribution of updates.

In this paper we rethink the way name resolution works in the Internet by proposing *DNS++: a distributed system able to resolve any name to an IP address in a private manner and continually update interested clients with new information about a service*. DNS++ is built on the privacy-preserving content-based pub/sub solution presented in [6] and [7], which is applied to the context of dynamic service name resolution using a large-scale overlay network of untrusted third party *Broker*s, acting as forwarder nodes. Thanks to the additive properties of the Paillier Homomorphic Encryption (HE) cryptosystem [8], Brokers can compare the content of encrypted subscriptions and notifications, determine whether the service names therein match, and perform routing and forwarding decisions accordingly. This approach offers better performance than existing fully HE schemes [9] and, when used as in [6], it also implements a shift of computational overhead from decryption to encryption that reduces the time needed to perform homomorphic matching on the Brokers, and ultimately contributes to make the system's deployment feasible on a today's distributed infrastructure.

This paper is organised as follows. Section II provides an overview of the main DNS++ actors, of the Paillier cryptosystem and of the contributions of this paper beyond the state-of-the-art; the system design and its main workflow are presented in Section III; in Section IV, the performance of the current DNS++ implementation is assessed, while considerations on the actual large-scale system deployment are made in Section V; some conclusions and potential future work are finally discussed in Section VI.

II. Background and Related Work

A. System Overview and Trust Model

Through the DNS++ naming system, a *Service Provider* can publicly advertise the availability of a *service* they offer, deployed via different *Service Replica*s on a set of resources. A service is accessible by potential *Client*s through an identifier — a Fully Qualified Domain Name (FQDN) [1] or DNS Uniform Resource Identifier (URI) [10] — mapped to each of the multiple replicas that implement that service. A Client wishes to select the best Service Replica for a given service according to some requirements (e.g., delay to the replica, server load, etc.), without revealing the details of that service to the name resolution system. Hence, when a Client specifies the identifier of the service of interest, together with the above requirements, a subset of all the possible Service Replicas will be returned, and the Client will be notified of relevant future updates related to that service. In order to support the above mechanisms, DNS++ is designed as a pub/sub system where Clients can issue queries about a service, receive the result of the resolution of that query, and subscribe to all the subsequent related updates. Service Providers push updates about replicas availability into the system. An Event Notification Service (ENS), consisting of an overlay network of *Broker*s, provides the mechanisms whereby each notification is forwarded from the Service Replica where it is generated to the Clients where it has to be notified.

Threats are considered in DNS++ from the point of view of Clients and Service Replicas with respect to third-party Brokers. We assume it would not be feasible to deploy an entire private network of Brokers under the ownership of a single entity when deploying the pub/sub system, hence an overlay of third-party Brokers is required. This poses some security issues, as those Brokers might not be trusted for the confidentiality of the content flowing through them.

Our system aims to ensure the privacy of Clients while they access (and subscribe) to specific services, as well as the confidentiality of publications related to Service Replicas. In other words, Brokers cannot identify the content of either the messages published by the Service Replicas or the service subscriptions issued by the Clients. This is achieved in our system via the usage of a homomorphic cryptosystem. We assume that Brokers are honest but curious, i.e., they perform the above protocol correctly but are curious to know what Service Providers publish and Clients consume.

B. Contributions beyond the state-of-the-art

DNSSEC [2] can only prevent attackers from manipulating or poisoning the responses to DNS requests. DNS over HTTP [3] and the work in [4] require Client queries to be processed by third parties, who are then able to view a query content; a PIR scheme is used in [5] to address this privacy issue. All these solutions do not support the dynamic distribution of service updates, which has been handled by previous work in [11], [12] instead. DNS++ aims to bridge the above gap by implementing a private name resolution system with dynamic distribution of updates; specifically, it goes beyond the current DNS query-response approach and uses pub/sub mechanisms, as discussed next.

A content-based network is a type of *pub/sub* where messages are forwarded hop-by-hop and delivered to any and all hosts that have expressed interest in the message content. DNS++ is similar to existing work on this topic [12]–[14] but it uses homomorphically encrypted service names and associated plaintext metadata, i.e., Attribute Value Pairs (AVPs), for the delivery of service updates to Clients. Although the work presented in [6], [7] builds a privacy-preserving content-based pub/sub system, the devised solutions are not specifically related to dynamic service name resolution, and do not consider how the proposed security protocols would work with a large overlay network of Brokers. This is in fact explored in our paper when a large-scale distributed content-based pub/sub is considered to enable dynamic service name resolution with built-in privacy.

Whilst the *encryption* of notifications provides the clear benefit of protecting the exchanged events, it also implies the inability of executing queries on their content and performing forwarding decisions accordingly. Searchable Data Encryption [15] and Homomorphic Encryption techniques [16] allow the execution of operations on the encrypted data without the need of performing any prior decryption. Specifically, HE enables limited computation to be performed directly on ciphertexts to generate encrypted results. *Partially* homomorphic cryptosystems, such as Paillier [8] and ElGamal [17], support the execution of a single type of arithmetic operation — either multiplication or addition — on encrypted data; *fully* homomorphic cryptosystem, such as BFV [18] and CKKS [19], allow both additions and multiplications to be performed on ciphertext values.

Like in [6], [7], a version of Paillier is utilised here to calculate an encrypted difference between values within subscriptions and notifications. It is used by untrusted third-party Brokers to perform homomorphic string match between service names within notifications and subscriptions, and to make routing and forwarding decisions accordingly. This approach offers better performance when compared to fully HE cryptosystems [9] and allows to reduce further the time required to perform homomorphic string matching on the Brokers. The details of the homomorphic cryptosystem that underpins DNS++ are provided in the next section.

C. Paillier Homomorphic Encryption

The security mechanisms discussed in this paper are based on a modified version of the Paillier cryptosystem originally presented in [8], where the tuple (λ, μ) is the *private key* and the tuple (n, g_p) is the *public key*. However, it can be proved that μ does not need to be private since it is hard to decrypt an encrypted message by only knowing μ [6]. Hence, μ can be made public while achieving the same security guarantees as the unmodified Paillier cryptosystem — the new public and private keys become (n, g_p, μ) and λ respectively.

Like in [6], only those holding the private key can encrypt, whereas the decryption is performed via the public key. Because λ is now utilised during the encryption, the resulting associated computational complexity is higher than in the original Paillier. On the other hand, λ is no longer used for the decryption, hence the related computational cost is reduced. It should be noted that all HE operations with the original Paillier cryptosystem are still available, as this modification only shifts computational overhead from decryption to encryption.

Confidential information is encrypted in a special way — called *blinding* — using the private key. Blinded values are semantically secure because two blinding operations performed on the same plaintext produce different blinded values. The blinding operation, which we indicate as *b = E(p)*, is devised so that the original plaintext p cannot be obtained by solely decrypting the blinded value *b* with the public key. However, when two blinded values are multiplied, some of the blinding parameters cancel out due to the HE properties of Paillier. A randomised difference between the original values is obtained via decryption of the product. It is utilised here, as in [6], to determine whether $E(p_1) \geq E(p_2)$ or $E(p_1) < E(p_2)$, without learning the original plaintext values p_1 and p_2. This property is used in our system to implement HE operations on blinded service subscriptions submitted by the Clients and blinded notifications about Service Replicas.

Brokers use the public key and the above blinded entries to perform two HE operations, which as in [6] are here referred to as *cover* and *match*. Given a publication pub and a subscription sub, *match(pub, sub)* allows to compare their blinded content homomorphically, and to determine the alphabetical order, or the equality, between the original s_n and s plaintexts; *cover(sub*$_1$*, sub*$_2$*)* provides similar functionalities but it operates on two subscriptions sub_1 and sub_2. The *cover* is used at routing time to build forwarding tables and to check whether a subscription is already included (covered) by an entry in the tables. The *match* is used at forwarding time and allows Brokers to check whether a notification they received matches one of the entries in their tables.

The execution of equality checks for *match* and *cover* operations is performed via verification of two separate inequalities. Specifically, to check whether $v_1 = v_2$, it is required to verify that $v_1 \geq v_2$ and $v_1 < v_2 + 1$. As a result, two blinded values $b_m = E(s)$ and $b1_m = E(s+1)$ are included within a subscription for a service name s; a notification only includes a blinded value $b_{m_n} = E(s_n)$ for a service name s_n associated with a replica. It should be noted that before a blind operation is performed, the string representing a service name is first converted to a bitstream using a standard character encoding, such as UTF-8.

The values b_m and $b1_m$ within subscriptions, and the value b_{m_n} within publications, are blinded so that when they are used for a *match* some of the blinding parameters cancel out and a randomised difference between the original plaintexts is obtained. Specifically, a Broker calculates $d = D(b_{m_n} \cdot b_m)$ and $d1 = D(b_{m_n} \cdot b1_m)$, where the public key is used to perform the decryption $D(\cdot)$. The randomised difference values d and $d1$ are utilised to perform the equality check for the service name, i.e., to determine whether $b_{m_n} \geq b_m$ and $b_{m_n} < b1_m$ [7]. A similar approach is used for the *cover*, hence an additional blinded value b_c is generated and added to the subscription by the Client. b_c is encoded so that the associated blinding parameters cancel out when it is multiplied homomorphically by either b_m or $b1_m$.

Public and private keys need to be carefully distributed among Clients, Brokers and Service Providers so that the confidentiality and privacy can be assured when performing the above HE operations. The *HE Parameters Service (HEPS)* generates and distributes the public parameters to relevant actors [6]. To minimise potential security threats due to the leakage of private parameters, blinding operations are carried out by the HEPS on behalf of the Service Providers. Clients are instead allowed to perform *self-blinding* of their queries via additional security parameters, which are generated for each of them by the HEPS using its own secret key. Clients are able to self-blind queries they are not required to interact with the HEPS for each query, thereby reducing the complexity and time taken to create and issue queries.

III. System Design

This section presents the design of the DNS++ system and describes its function through a workflow consisting of four main steps. In *step 1*, the Brokers are deployed on the resource infrastructure and interconnected as an overlay network; in *step 2*, as the Clients start to subscribe to services, forwarding tables are populated in the Brokers; in *step 3*, updates about Service Replicas are generated; finally, in *step 4* the service names within the updates are compared homomorphically with the content of the forwarding tables, so that replicas information is propagated to the Clients accordingly.

A. STEP 1: Overlay initialisation

The ENS consists of a set of Brokers interconnected as an overlay. Although we do not prescribe the number of Brokers nor the degree of interconnectivity between them, as a starting point we assume that each Autonomous System (AS) will run a Broker, and that a Broker will connect to the Brokers of adjacent ASs following the underlying interdomain network topology. We also assume that Clients connect to the local Broker of their ISP's AS. A simple example scenario is is represented in Figure 1. For generality, communications between

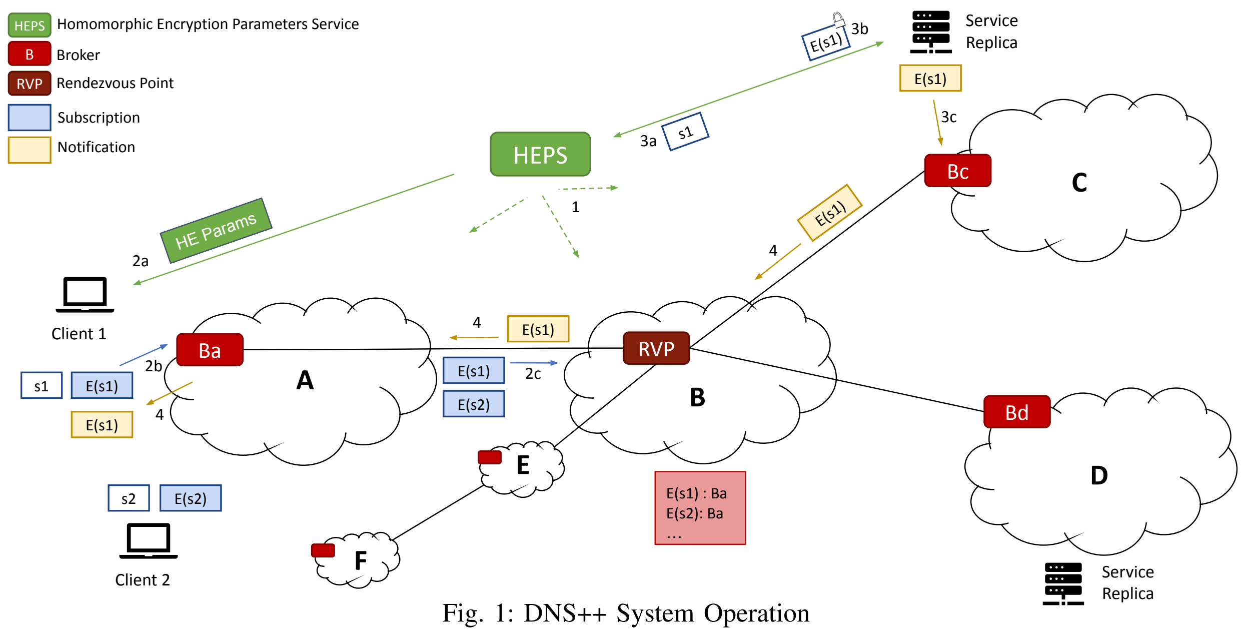


Fig. 1: DNS++ System Operation

Brokers will happen through the overlay's application-layer protocol rather than through BGP.

To avoid all subscriptions and updates needing to be propagated to every Broker in the overlay, one Broker is nominated as the *Rendezvous Point* (RVP) for each service [14]. The same Broker will act as a RVP for many services. The RVP is the default Broker where subscriptions and updates are matched; however, in the majority of cases matching will be achieved in Brokers other than the RVP. To comply with our principles of privacy and anonymity it should not be possible to reverse engineer the identity of a service during the name resolution process, which includes being unable to identify the service given its RVP.

We use a shared colliding hash function is used to map a service name to its corresponding hash, where the *N_of_hashes* $\ll$ *N_of_names* so that the service name can not be inferred from its hash. Each hash resolves to a single Broker, which is the RVP for the hashed service name (and, in fact, the RVP for all service names sharing the same hash value). In Figure 1 for instance, the Broker in domain B acts as the RVP for both service *s1* and service *s2*. Brokers announce the hashes for which they are responsible to their neighbouring Brokers who, in turn, propagate hash reachability information to their neighbours, and so on, to construct forwarding tables for hashes throughout the broker overlay. Clients, when subscribing to service names, and publishers, when announcing updates, include the plaintext hash of their service name with the subscription/update. The hash acts as the identifier for the RVP for that service. Brokers, when forwarding subscriptions and updates to the RVP identify the next hop Broker by using the hash as an index into the forwarding tables constructed in the overlay initialisation phase.

The final stage of overlay initialisation is for the Brokers to contact the HEPS to retrieve the public parameters necessary for calculating *cover* and *match* operations — this is represented in Figure 1 as *step 1*.

B. STEP 2: Subscription

A Client that wishes to query/subscribe for service updates interacts with the HEPS to retrieve a set of security parameters they will use to perform self-blinding operations (*step 2a*). The Client generates the blinded values b_m and $b1_m$ that Brokers will use for the *match* operation, as well as the value b_c required for the *cover*. The values are then added to the subscription, together with any AVPs that describe the service requirements and the hash of the service name, which identifies its RVP. The Client also adds a subscription identifier to the header, which is used to distinguish between currently active subscriptions generated by that Client, so that incoming notifications can be associated with the correct subscription. The ID does not need to be globally unique and it will not identify the Client. Finally, the complete subscription is sent by the Client to their local Broker (*step 2b*).

As Brokers process homomorphically the content of the received subscriptions, they can build their forwarding tables without learning the service names. To minimise table inflation, subscriptions related to the same service, received from the same overlay link, should be discarded. Hence, Brokers use the public key provided by the HEPS to perform *cover* operations, namely to check homomorphically whether a received subscription is already in their forwarding table. The *cover* operation additionally allows to determine the alphabetical order of subscriptions being checked. This helps organising the tables as sorted data structures (e.g., self-balancing binary search trees), which may result in more efficient *match* operations at forwarding time.

If a new subscription is not covered by an existing one, then the Broker first of all creates a new local forwarding entry for the subscriber, so that future updates for that service name can be routed to the subscriber via the overlay link it received the subscription over; and, secondly, it forwards the subscription towards the RVP. To do the latter it uses the plaintext hash of the service name provided by the Client in the subscription to retrieve the next hop Broker from the RVP forwarding table constructed during the overlay initialisation phase. If a new subscription is covered by an existing subscription, but the forwarding table for the subscription does not include the link from which the new subscription arrived, then the Broker adds the link to the forwarding table for future updates.

C. STEP 3: Notification

A Service Provider that wishes to announce or update the information about a Service Replica first authenticates with the HEPS and provides the plaintext s_n of the service name to be blinded (*step 3a*). The HEPS verifies the credentials of the Service Provider, calculates the requested blinded value b_{m_m} and adds its digital signature to the generated data. Next, any additional AVPs that describe the features of the Service Replica are added to the notification by the Service Provider together with the hash of the service name to identify the RVP for the notification. Finally, the notification is sent to the Service Replica's local Broker (*step 3b*).

D. STEP 4: Match

When a new notification is received by a Broker, the validity of its content is first checked by verifying the digital signature added by the HEPS in step 3. Then, the blinded service name b_{m_n}, received as part of the notification, is homomorphically checked for equality with the blinded service names of the subscriptions within the local forwarding table. This is done by performing a *match* operation using the public key received from the HEPS. Specifically, a Broker invokes *match(pub, sub$_i$)*, where *pub* is the received publication and *sub$_i$* is the *i-th* subscription in its forwarding table. Thanks to the alphabetical comparison properties of the *match* and the ordering of the entries within the forwarding table, the number of invocations of the above *match*, required to explore the table exhaustively, is bounded by $log_2(N_s)$, where N_s is the number of entries in the table.

If two service names match, then additional checks are performed on the other plaintext AVPs to determine whether the attributes associated with the Service Replica in the notification fulfil the criteria specified in the matching subscriptions (note that this is not explicitly represented in Figure 1 to simplify the diagram). Matching notifications are then sent over the overlay links contained in the forwarding tables and eventually to the subscribers (Clients). This is shown in Figure 1 by the arrows labelled as *step 4*. When a notification is forwarded to a Client the subscription ID that was supplied by the Client (and was maintained in the forwarding table entry for that Client by the local Broker) is added to the notification. In addition to forwarding notifications towards Brokers and Clients that match any existing subscriptions, the Broker also forwards the notification towards the RVP for the service. This uses the hash of the service name as provided in the notification to look up the next hop Broker from the RVP forwarding table.

It should be noted that our system allows for Client queries to be resolved immediately as well as for Clients to subscribe to future updates. To achieve this, each Broker caches the most recent notification it has received from each replica of a service name. When a new subscription is received, and it is covered by an existing one, then the cache of the prior notifications is returned immediately to the Client [20].

IV. Evaluation

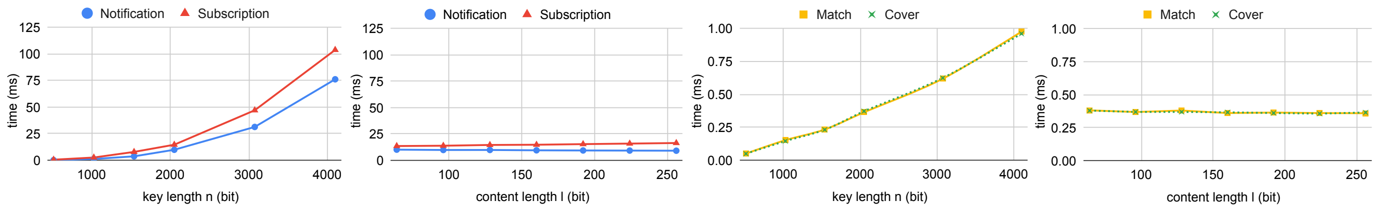
A proof of concept implementation of DNS++ was utilised for the preliminary evaluation discussed in this section[1]. Brokers' forwarding tables were implemented as homomorphic encrypted self-balancing binary search trees for fast storage and retrieval of ordered service names information. Tests were performed on a server with an Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz and 192GB of RAM to assess the impact of homomorphic *match* and *cover* operations on the ENS, as well as the time required to generate service subscriptions and notifications.

Subscriptions and publications were initially produced via randomly generating content for the service names and evaluating the impact of both the key length n and the number of bits l utilised to represent the service name. It should be noted that, as in [6], l is assumed to be sufficiently smaller than n. The graphs of Figure 2a and Figure 2c show the time required to generate the blinded values for both subscriptions and notifications, and to execute *match* and *cover* operations, when the number of bits $l = 128$ and various key length values n are considered. Measurements related to the same operations are shown in the graphs of Figure 2b and Figure 2d for a fixed value of the key length ($n = 2048$) and different values of the parameter l. The average of the results, calculated over 1000 iterations of an experiment executed under the same parameters settings, is represented in the graphs; confidence intervals were negligible and are not shown.

The time measured for the generation of either subscriptions or notifications grows considerably with the length of the key n (Figure 2a) while it is not impacted by the number of bits l utilised for encoding the service name (Figure 2b). Likewise, the execution time of either a *match* or *cover* operation (Figure 2c) also increases with the size of the key but is not impacted by the value of the parameter l (Figure 2d).

It should be noted that the computational shift from decryption to encryption, implemented by the modified version of the Paillier cryptosystem, can effectively reduce the time associated to the execution of *match* or *cover* operations when compared to the generation of publications or subscriptions. This fits well with a pub/sub scenario, as it removes the more computationally intensive operations from the Brokers at both routing and forwarding time, resulting in a potentially lower end-to-end latency of the ENS.

[1]The Java code is available at https://github.com/francesco-tusa/dnsPlus



(a) *blind*: variable n (b) *blind*: $n = 2048$, variable l (c) *match* / *cover*: variable n (d) *match* / *cover*: $n = 2048$, variable l

Fig. 2: Results of the preliminary DNS++ performance evaluation

For all the key length values n evaluated during the experiments, the graphs confirm that the time required to perform a *match* (or *cover*) is always smaller than the time needed for generating either a subscription or a publication. In particular, when $n = 2048$, performing a *match* (or *cover*) requires approx one third of the time it takes to produce a publication; moreover, creating a subscription is consistently slower than producing a publication. This is an expected result, as a publication includes a single blinded value b_{m_n} whereas, for a subscription, both the blinded values b_m and $b1_m$ need to be encoded — together with the blinded value b_c required for the *cover* — to allow Brokers to determine the alphabetical order between the blinded service names. A *match* always requires verification of the inequality $b_{m_n} \geq b_m$; the second inequality $b_{m_n} < b1_m$ only needs to be checked if the first one is true. Execution of *cover* is similar but the value b_c is used in place of b_{m_n}. In both cases, two homomorphic multiplications and the decryption of the obtained results need to be computed in the worst case, leading to comparable performance when those operations are executed, as shown in Figures 2c and 2d.

To assess the performance of DNS++ on a Broker deployed on a typical domain, we downloaded a list of the 1000 most popular websites from [21], which act as 1000 service names. This input was used to run the protocol, considering a key length $n = 2048$ bits and a content information size $l = 256$ bits. After the forwarding table was built, six different notifications were generated, five of which referred to services with different degree of popularity randomly selected from the above list of websites. The average time required to *match* each of those publications was measured and the results of one of these tests are reported in Table I.

Service Name	**Match Time** (ms)
allmusic.com	3.97
google.com	2.32
facebook.com	0.45
⟨not in the table⟩	3.49
ticketmaster.com	4.48
eventbrite.co.uk	3.91

TABLE I: Time required to perform service *matches*

From the collected measurements, it was calculated that the average time required to check whether a publication *match*ed any of the N_s subscriptions in a Broker's forwarding table was 3.1ms, which is consistent with the expected $O(log_2(N_s))$. It should be noted that, although this performance evaluation was based on just 1000 service names, the logarithmic nature of both insert and match operations in a binary tree means that one billion names — which is three times the current amount of domain names registered globally [22] — would only increase the time of table creation and match operations by a factor of three, making the average match time 9.3ms in the worst case.

V. Deployment Considerations

In DNS++, replicas should only be announced to "local" clients, i.e., announcements should not traverse the whole overlay and reach distant clients as these may have access to existing replicas nearby that already satisfy their requirements. This can be done in a myriad of ways by using the plaintext AVPs of the notifications. Examples include prefix limitation where replicas announce IPv4/IPv6 prefixes of the areas they want to cover; use of network coordinates; explicit naming of AS numbers; etc. The fact that RVPs may fall outside these areas presents a problem but there are solutions from previous work [23] that address this.

We assumed earlier the existence of a single Broker per AS and that Brokers are interconnected following the inter-provider AS topology. However, there could be a finer-grained set of Brokers, e.g., more than one for large ASs, or a coarser-grained set of Brokers that are not tied to any specific AS and are operated by entities other than ISPs. Also the degree of interconnection between Brokers does not necessarily need to follow that of BGP peers. For instance, it may be more efficient for Brokers to connect directly to distant Brokers. The degree of connectivity implies a trade-off between efficiency, in terms of the size of forwarding tables maintained in Brokers, and the computational complexity of match operations on larger tables caused by higher degrees of connectivity. The design of Broker overlay discovery and construction mechanisms can be built upon solutions that have already been widely discussed in the literature, e.g., [24]–[26], and to which we have partly referred to in order to divide the hash space for RVPs.

Our design does not constrain the conceptual structure of the HEPS nor its implementation/deployment. Trusted organisations that already act as CAs today, as well as Internet registries, are good candidates to implement HEPS functionality. Clients need to contact the HEPS only once and can subsequently perform self-blinding of service names for all future queries without needing further interaction with the HEPS. Service Providers have to contact the HEPS when they wish to blind a new service name however, after this has been encoded and signed by the HEPS, they are able to generate updates for all Service Replicas related to that name. This implies no significant overhead above that required for registering a domain name today. There are around 10 million new DNS name registrations per year [22], which implies an average of one signing operation every three seconds globally.

VI. Conclusions

This paper presented DNS++, a system designed to address both the lack of privacy and dynamic distribution of service information of the current Internet's name resolution system. A pub/sub overlay underpins the DNS++ architecture so that, as the features of services offered by Service Providers dynamically change, relevant updates are continuously delivered to interested Clients, who can (re)select between available Service Replicas according to their requirements. The system uses homomorphic encryption to prevent untrusted parties from accessing encrypted service information. Client privacy is preserved as Brokers can only perform homomorphic name match and forward service updates to relevant Clients through the overlay accordingly, without being able to access the original plaintext values.

When forwarding tables in each Broker are implemented via ordered data structures the time required for adding a new Client's subscription to a table, and to match existing subscriptions with service updates with HE operations , grows logarithmically with the number of entries within that table. Thanks to the computational shift implemented by the chosen Paillier scheme [6], homomorphic *match* and *cover* operations can be performed with acceptable performance overhead when today's 2048 bits minimum recommended key length for this type of cryptosystem is considered [27].

Ongoing work includes the investigation of the localisation aspects of the system in order to limit the announcement of replica updates to specific clients based on geographic constraints. Overlays with different topologies are also being evaluated with regard to their impact on the size of forwarding tables in Brokers, as well as on the number of *cover* and *match* operations performed at routing and forwarding time in the overlay as a whole.

VII. Acknowledgements

The authors would like to acknowledge the support of Huawei Technologies Co., Ltd. Francesco Tusa is partially supported by the HARPOCRATES EU research project (No. 101069535).

References

[1] "Domain names — implementation and specification." RFC 1035, Nov. 1987.

[2] D. E. E. 3rd, "Domain Name System Security Extensions." RFC 2535, Mar. 1999.

[3] N. P. Hoang, I. Lin, S. Ghavamnia, and M. Polychronakis, "K-resolver: towards decentralizing encrypted dns resolution," *arXiv preprint arXiv:2001.08901*, 2020.

[4] A. Hounsel, P. Schmitt, K. Borgolte, and N. Feamster, "Encryption without centralization: distributing dns queries across recursive resolvers," in *Proceedings of the Applied Networking Research Workshop*, pp. 62–68, 2021.

[5] N. Dokmai, L. J. Camp, and R. Henry, "Assisted Private Information Retrieval." Cryptology ePrint Archive, Paper 2022/1082, 2022. https://eprint.iacr.org/2022/1082.

[6] M. Nabeel, N. Shang, and E. Bertino, "Efficient privacy preserving content based publish subscribe systems," *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT*, 06 2012.

[7] M. Nabeel, S. Appel, E. Bertino, and A. Buchmann, "Privacy Preserving Context Aware Publish Subscribe Systems," in *Network and System Security* (J. Lopez, X. Huang, and R. Sandhu, eds.), (Berlin, Heidelberg), pp. 465–478, Springer Berlin Heidelberg, 2013.

[8] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology — EUROCRYPT '99* (J. Stern, ed.), (Berlin, Heidelberg), pp. 223–238, Springer Berlin Heidelberg, 1999.

[9] V. Sidorov, E. Y. F. Wei, and W. K. Ng, "Comprehensive Performance Analysis of Homomorphic Cryptosystems for Practical Data Processing," 2022.

[10] S. Josefsson, "Domain Name System Uniform Resource Identifiers." RFC 4501, May 2006.

[11] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav, "A global name service for a highly mobile internetwork," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 247–258, 2014.

[12] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, SOSP '99, (New York, NY, USA), p. 186–201, Association for Computing Machinery, 1999.

[13] A. Carzaniga, M. Rutherford, and A. Wolf, "A routing scheme for content-based networking," in *IEEE INFOCOM 2004*, vol. 2, pp. 918–928 vol.2, 2004.

[14] P. Pietzuch and J. Bacon, "Hermes: a distributed event-based middleware architecture," in *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, pp. 611–618, 2002.

[15] C. Dong, G. Russello, and N. Dulay, "Shared and Searchable Encrypted Data for Untrusted Servers," in *Proceeedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, (Berlin, Heidelberg), p. 127–143, Springer-Verlag, 2008.

[16] X. Yi, R. Paulet, and E. Bertino, *Homomorphic Encryption*, pp. 27–46. Cham: Springer International Publishing, 2014.

[17] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[18] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[19] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Advances in Cryptology – ASIACRYPT 2017* (T. Takagi and T. Peyrin, eds.), (Cham), pp. 409–437, Springer International Publishing, 2017.

[20] "ZeroMQ guide, Advanced Pub-Sub Patterns: Last Value Caching (LVC)." https://zguide.zeromq.org/docs/chapter5/#Last-Value-Caching.

[21] "List of the 1000 most popular websites." https://gist.github.com/bejaneps/ba8d8eed85b0c289a05c750b3d825f61.

[22] "The Domain Name Industry Brief." https://www.verisign.com/en_US/domain-names/dnib/index.xhtml.

[23] J. Hasenburg and D. Bermbach, "Using geo-context information for efficient rendezvous-based routing in publish/subscribe systems," *KuVS-Fachgespräch Fog Computing 2020*, p. 4, 2020.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 161–172, 2001.

[25] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on networking*, vol. 11, no. 1, pp. 17–32, 2003.

[26] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[27] "BlueCrypt: Cryptographic Keylength Recommendation." https://www.keylength.com/en/compare/.