CFRG
Internet-Draft
Intended status: Experimental
Expires: 14 July 2026

C-DAC Pune
C-DAC

10 January 2026

Verifiable Delay Tokens for Privacy-Preserving Time Enforcement
draft-bakshi-vdt-verifiable-delay-token-00

Abstract

This document specifies a protocol for the issuance and verification of Verifiable Delay Tokens (VDT) that proves that a minimum amount of wall time has elapsed since the issuance of the VDT. The protocol does not rely on trusted or synchronized time sources and instead uses Verifiable Delay Functions (VDFs). The protocol supports time-based access control, anti-abuse and privacy-preserving mechanisms by letting the verifier enforce a waiting period, mitigating the abuse without learning client identity. The protocol ensures unlinkability between token issuance and redemption and does not require the verifier to be aware of the client identifiers, network addresses or any long-lived state. VDTs compose well with the existing Internet protocols such as HTTP-based systems and Oblivious HTTP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 14 July 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

Many Internet services require time-based access control to enforce rate limits, mitigate abuse or require clients to wait before retrying access to a resource. For example, after repeated failed authentications (HTTP 401), too many requests (HTTP 429) and challenge-based defenses (CAPTCHA) against automated abuse by bots.

Typical existing mitigation techniques require the following from clients and verifier.

* Let verifier identify clients across multiple requests.

* Let verifier maintain state of the clients.

* Let client trust the clock maintained by verifier.

* Let the logical connection between client and verifier be maintained during the delay.

These requirements introduce privacy challenges. The ability of verifier to identify clients across multiple requests enables linkability of multiple requests and trackability of client behavior over time. Maintaining state of the clients at verifier enables correlation across requests and increases complexity. The full reliance on the clock maintained by verifier can lead to time manipulation by verifier. A continuous logical connection requires continuous stable connection and increases the verifier load and the latency.

Hence, at present, there is a lack of a standardized mechanism to let a client prove to a verifier that a minimum amount of real time has elapsed without revealing client identity, without relying on trusted clock and without maintaining continuous logical connection with the verifier during the delay period.

This document aims to address this gap.

1.2. Verifiable Delay Functions as a Time Primitive

A Verifiable Delay Function (VDF) is a cryptographic primitive that requires a prescribed sequential time for the computation even with massive parallelism and produces a result which can be verified efficiently. The VDF properties makes it apt for enforcing elapsed real time.

* Sequentiality: The computation time relies on the elapsed real time rather than the computational power.

* Efficient Verifiability: Any verifier can verify the correctness of the result efficiently.

* Clock Indepdendence: Time sources need not be synchronized or trusted.

Note that VDFs enable a bound on elapsed time rather than computational power which makes it different from the Proof-of-Work constructions.

1.3. Overview of Verifiable Delay Tokens

Verifiable Delay Token (VDT) aims to bind the challenge issued by an issuer to the result of the VDF and later be presented to a verifier as a proof of the elapsed time. At a broad level, the protocol works as follows.

1. A client requests a delay challenge from an issuer without revealing its identity.

2. The issuer provides VDF parameters and a challenge value without tracking the client.

3. The client computes the VDF without the need of being online or to interact.

4. The client constructs the VDT and presents it to the verifier.

5. The verifier efficiently verifies that the required delay has elapsed without requiring to re-compute the VDF, without trusting the timestamps and without the need to know when the computation was started.

The protocol ensures the following.

1. The issuer does not learn who redeemed the token, at what time and using which of the issued challenges.

2. The verifier does not learn when and to whom the token was issued.

3. The client does not reveal a identifier during issuance and redemption of the token.

Note that issuer and verifier MAY be distinct entities enabling deployment models which preserves unlinkability between issuance and redemption.

1.4. Design Goals

The protocol aims to meet the following design goals.

* Elapsed time enforcement: Ensure that a minimum amount of elapsed real time.

* Privacy Preservation: Ensure unlinkability between issuance and redemption by avoiding inclusion of client identifiers.

* Zero-trust on Clock Sources: Avoid any trust on the synchronized or trusted clock sources.

* Offline Delay Computation: Let delay computation be done without the need of an online interaction.

* Efficient Verification: Ensure that the verifier be able to verify the token efficiently.

* Composability: Let the integration of the protocol with the existing Internet protocols be easy.

1.5. Non-Goals

Following details are intentionally kept of scope from this document.

* Who the client is, how it is authenticated and what it is authorized to do.

* What are the fees for tokens, economic incentives, pricing or payment mechanisms.

* What, if any, hardware attestation or trusted execution environments are used.

* Fairness guarantees across heterogenous client devices such as a mobile phone, a laptop or a GPU cluster.

* Application-specific access control policies.

1.6. Applicability

VDTs are intended to be used in scenarios where a verifier wants to enforce a time delay without the need of tracking clients or to maintain persistent state. Some of the applications are listed below.

* Privacy-preserving rate limiting such as permit 100 requests per IP per hour.

* Challenge-response systems without CAPTCHAs which could have otherwise track users.

* Deferred access to sensitive resources even if access is allowed such as in case of repeated password reset attempts.

* Abuse mitigation in anonymous or oblivious transport systems in which the traditional defense mechanisms fail because of anonymized client identities.

1.7. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are used throughout this document.

* Verifiable Delay Function (VDF):

A VDF is a cryptographic function that requires a certain sequential computation to complete, making it unable to parallelize the computation effectively while yet enabling efficient verification of the outcome using publicly available data. A VDF is defined by algorithms for setup, evaluation, and verification and by a parameter that controls the desired delay.

* Verifiable Delay Token (VDT):

A VDT is a cryptographic token created from the result and proof produced by a VDF computation, and the metadata necessary for verification. A valid VDT serves as a proof that a minimum amount of time has elapsed since the challenge issuance without revealing client identity or requiring trusted clocks.

* Client:

A Client is the entity that initiates a delay challenge, completes the VDF evaluation, and provides a VDT for validation. The client is not required to authenticate itself, be a trusted entity and reveal its identifier.

* Issuer:

An Issuer is the entity that issues delay challenges, VDF parameters and challenge seed. The Issuer MAY apply necessary policies to challenge issuance, is not involved in delay computation and is not required to learn about token redemption.

* Verifier:

A Verifier is an entity that validates VDT and determines whether the delay requirement has been met. A Verifier is not required to maintain a per-client state. The verification of the VDF computation given the proof and the output is an efficient operation.

* Relay:

A Relay is an optional entity that transfers protocol messages among Clients, Issuers and Verifiers. Relay does not have access to plaintext content of the message. Relays are used to provide network-level privacy and are assumed to be honest-but curious.

* Challenge:

A Challenge is a set of parameters issued by an Issuer in Challenge Response message. It defines the input and difficulty for a VDF computation. At minimum, a Challenge contains a challenge seed and a delay parameter.

* The Challenge Seed:

The Challenge Seed is a fresh, unpredictable value generated by the Issuer and used as input to the VDF. Challenge seeds MUST have a high probability of being unique and MUST NOT be used in more than one Challenge.

* Delay Parameter:

The Delay Parameter indicates the intended length of the VDF computation. It does not indicate an absolute value and its interpretation is VDF-specific.

* VDF Output:

The VDF Output is the deterministic value obtained by computing the VDF on a given input and delay parameter.

* VDF Proof:

VDF Proof is a cryptographic object that enables efficient verification of the VDF Output with respect to the input value and delay parameter.

* Context Binding:

Context Binding is an optional mechanism that binds a Verifier-defined scope, purpose or request such as reattempt post failed-authentication, resource identifier, a service domain to a VDT. If used, context binding is included in the VDF input.

* Validity Window:

A Validity Window is an optional constraint that restricts the usage of a VDT within time duration indicated by validity window. Validity windows do not require synchronized clocks and are used to reduce replay attack risk.

* Unlinkability:

Unlinkability is the inability of an observer to correlate protocol events such as Challenge Issuance and Token Redemption to the same Client.

* Replay:

A Replay of a VDT occurs when the same VDT which was used and accepted earlier to obtain access is used again to obtain the access again.

* Stateless Verification:

Stateless Verification is the verification that does not require the Verifier to keep long-lived session or per-client state beyond what is required to prevent token reuse.

* Domain Separation:

A Domain Separation is a cryptographic technique used to ensure that inputs and outputs from different protocols or contexts cannot be reused across protocol boundaries.

2. Background and Design Rationale

This section presents the existing mechanisms for time-based access control, why they are not sufficient to meet the goals mentioned in Section 1 and how VDF can be useful as the basis of the protocol.

2.1. Server-Enforced Timers

One of the existing approaches to implement time-based access control is to let the server trust the timestamp in the Client's request and maintain a state with the Client. The server accepts the Client's request only if it is at least required time duration later than the earlier request from the same client. Such mechanisms have several privacy limitations. Maintaining a Client state across multiple requests creates Client linkability across multiple requests typically through IP addresses, cookies or authenticated sessions. This requires server to maintain per-client state for at least the duration of the delay interval. This also requires the server to trust the clock source of the server's time source. A server can easily correlate a Challenge Request message with the corresponding Challenge Response message. A server enforced timer may be acceptable in deployments where revealing client identification is acceptable, however, it is incompatible in deployments where privacy-preserving and anonymous access control is a requirement.

2.2. Client-Side Timers and Delayed Requests

Other approach is to let the server trust the Client in which case the Server will simply accept the request received from Client assuming it is sent only after a requisite time delay. In this case, a malicious Client can easily reduce or avoid the requisite delay unless a trusted Client hardware or server-side verification is used.

2.3. Proof-of-Work Mechanisms

Proof-of-Work (PoW) schemes require Clients to perform a specific computational work to gain access to a resource. The time to compute PoW is related to the computational power and not the elapsed time. PoW is related to the computational power and not the elapsed time, can often be computed faster using parallel techniques, varies widely across heterogenous devices leading to unfairness and can impose unnecessary energy consumption. Hence, a PoW is not suitable for scenarios where minimum real-time delay is a requirement. This distinction between PoW and VDF is discussed in foundational work on VDFs [BONEH-VDF].

2.4. Token Expiry and Credential-Based Approaches

Another approach is to encode the expiry timestamp in the token itself. This approach requires Verifier to trust the time source of its clock and of the Client, and binds a token to an identifiable Client. Such mechanisms are not suitable for privacy-preserving unlinkable time-based access because a token is associated with an identifiable Client which can easily link between challenge issuance

and token redemption messages, and synchronization between Clients and Verifier.

2.5. Limitations of Existing Delay Mechanisms

Existing mechanisms are insufficient to ensure the enforcement of real-world elapsed time, absence of trusted time sources, absence of client identifier, stateless verification and offline delay computation. This document introduces a standardized mechanism which satisfies all of these requirements simultaneously.

2.6. Verifiable Delay Functions as a Building Block

VDFs provide a cryptographic mechanism for enforcing elapsed time through sequential computation such that an adversary cannot compute significantly faster even with massive parallelism. VDFs have some nice properties which are useful for the protocol. The sequential computation of the VDF is directly proportional to the delay parameter. The verification of the VDF is an efficient operation and is significantly faster compared to its computation. The output of the VDF is uniquely determined by its input and parameters. VDF does not require any trusted or synchronized time clock.

2.7. Tokens Derived from Delay Proofs

The VDTs introduced in this document are cryptographic objects derived from the computations of the VDF. Binding VDF proofs to the tokens enables stateless verification, unlinkability between challenge issuance and token redemption and composition with existing Internet protocols. The abstraction of a VDT over VDF enables Verifiers to find whether the required time duration has elapsed without requiring to know when, where or how the VDF computation was done.

2.8. Separation of Issuer and Verifier

The protocol allows the Issuer which issues the challenge and the Verifier which verifies the token derived from that challenge to be separate entities. This separation is intentional to enable privacy-preserving deployments where the Issuer is not required to know when or where a token is redeemed and the Verifier is not required to know when or under which condition the corresponding challenge was issued.

3. Assumptions, Threat Model and Goals

This section presents the assumptions, the threat model and the security and privacy goals.

3.1. Trust and Privacy Assumptions

The protocol makes the following trust assumptions.

* The Issuer is trusted to select delay parameters in accordance with the local policy and to generate fresh unpredictable challenge seeds.

* The Verifier is trusted to correctly verify the VDT and produce the verification results.

* The Relay, if used, is honest-but-curious and does not tamper protocol messages.

* The Client is not trusted and may try to avoid or shorten delay computation.

The protocol does not require synchronized clocks, trusted client hardware or trusted time sources.

The protocol makes the following privacy assumptions.

* No collusion between Issuer and Verifier.

* No inclusion of Client identifiers in the challenge requests and token redemption requests.

* Network metadata is suitably hidden by the oblivious transport or the optional Relays.

Any compromise in these assumptions can diminish the privacy guarantees.

3.2. Threat Model

The protocol assumes the presence of following adversaries.

* Malicious Client, which may attempt to forge a VDT, shorten the VDF computation time, replay the earlier VDT or redeem a VDT in the wrong context.

* Malicious Issuer, which may attempt to reuse challenge seeds, issue challenges with insufficient delay parameters or include identifying information into the challenge.

* Malicious Verifier, which may attempt to gain additional information that compromise Client’s privacy, correlate multiple token redemption requests or enforce incorrect access policies.

* Malicious Network Entities, which may observe, block, replay or delay protocol messages.

The protocol assumes the presence of secure network communication.

The protocol keeps the following threats out of scope.

* Collusion between the Issuer and the Verifier.

* Side-channel attacks on client hardware.

* Covert channels in Client’s delay computation.

* Denial-of-Service attacks that go beyond standard mitigation

* The enforcement of absolute wall-clock time guarantees.

3.3. Securities and Privacy Goals

The protocol aims to achieve the following goals under the stated assumptions.

* Manifest the minimum elapsed time through sequential computation.

* Prevent token forgery via fabricating a valid token or shortening the delay computation.

* Enable stateless verification.

* Preserve Client’s anonymity and unlinkability across multiple requests.

* Avoid reliance on local system time, synchronized clocks or Client identifiers.

4. VDT Protocol

This section presents the participants, their interactions, underlying assumptions, threat model and the security and privacy goals in the VDT protocol.

4.1. Participant Entities

The protocol involves the following entities.

* Client: Entity which for time-controlled access to a resource, requests a delay challenge from the Issuer, computes VDF, constructs VDT and presents the VDT to the Verifier.

* Issuer: Entity which generates and issues delay challenges and associated parameters to the Clients.

* Verifier: Entity which receives VDTs from Clients and determines whether the required delay has elapsed.

* Relay (Optional): Entity which forwards protocol messages among Clients, Issuers and Verifiers without requiring access to the plaintext content.

Note that the Issuer and the Verifier MAY be the same or the distinct entities. If distinct, the protocol supports unlinkability between the challenge issuance and the token redemption.

4.2. Protocol Phases

The protocol primarily consists of the three phases in sequence, Challenge Issuance, Delay Computation and Token Redemption. The Challenge Issuance phase does not require participation of the Verifier and the Delay Computation phase does not require participation of the Issuer. If a Relay is used, the messages MAY be forwarded by it without modification.

1. Challenge Issuance Phase

In this phase, the Client sends a Challenge Request to the Issuer and the Issuer responds with the Challenge Response to the Client. The Challenge Response contains a challenge seed, a delay parameter, VDF parameters and Issuer metadata. requests delay parameters from the Issuer. The Issuer MUST NOT include in Challenge Response, the information which can be used later to identify the Client at redemption. The Issuer MAY enforce suitable access policy such as rate-limitation but such policy is kept out of scope for this document. Figure 1 illustrates the Challenge Issuance phase. Figures in this document are illustrative and non-normative.

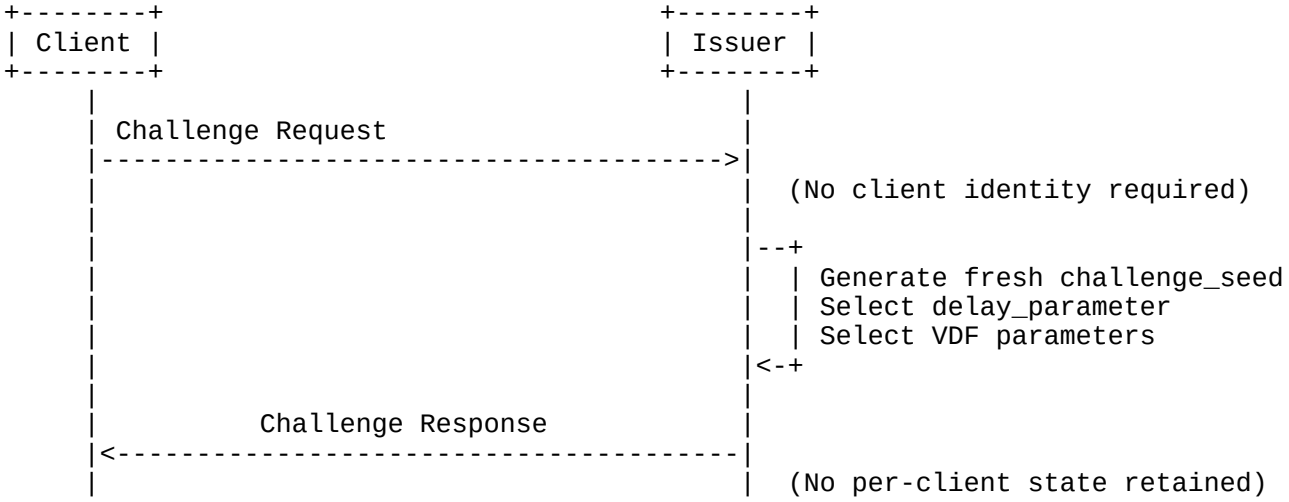

Figure 1: Challenge Issuance Phase

1. Delay Computation Phase

In this phase, the Client computes the VDF locally and offline using the delay parameter and VDF parameters received from the Issuer and produces a VDF output and a corresponding proof. The time taken for the computation of the VDF is proportional the delay parameter even with massive use of parallelism. A VDT is computed containing the challenge seed, the delay parameter, the VDF parameters, the VDF output and the VDF proof. For VDT Format refer section 7. Client MAY not be connected with the Issuer or the Verifier during this phase. Figure 2 illustrates the Delay Computation Phase.

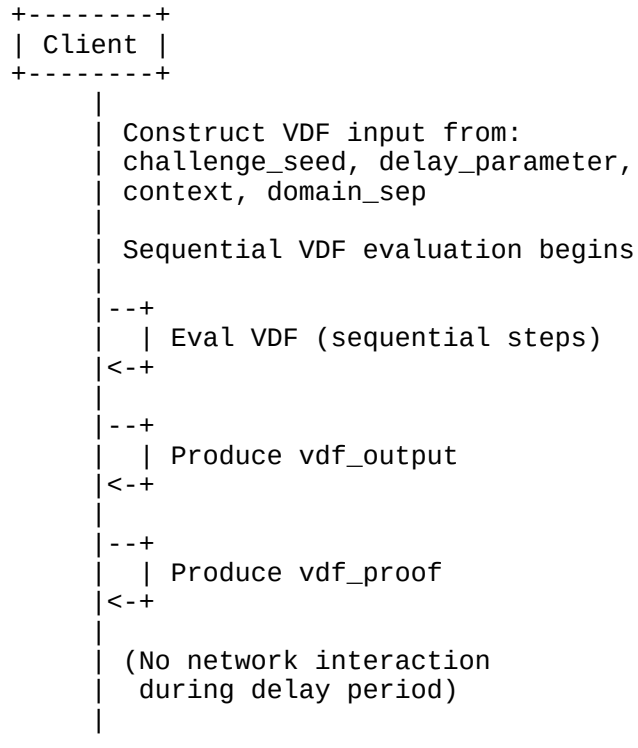

Figure 2: Delay Computation Phase

1. Token Redemption Phase

In this phase, the Client send a Token Redemption Request containing the VDT to a Verifier. The Verifier validates the token version, the Issuer, the VDF proof, the context, and the freshness included in the VDT. The Verifier returns a Verification Result which MAY grant access to a resource. The Client MUST NOT include information which can identify the Client. Figure 3 illustrates the token redemption phase.

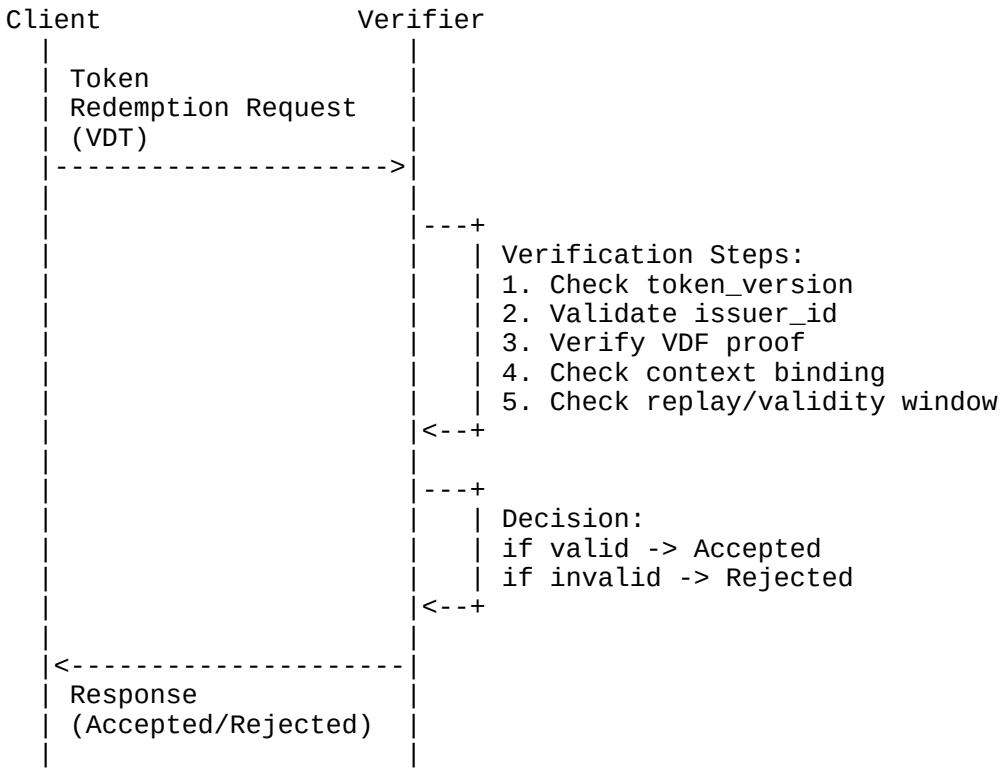

Figure 3: Token Redemption Phase

4.3. Binding Context with VDT

A VDT MAY optionally be associated with a context defined by the Verifier. The context is defined by the Verifier and shared with the Issuer out-of-band. Later, if required, the context is given to the Client by the Issuer in Challenge Issuance phase. Binding context to a VDT allows a Verifier to restrict usage of the token to a specific scope. If context is used, the Client MUST include the context value in the VDF input and the Verifier MUST reject tokens computed without the expected context. Note that binding context with token does not affect unlinkability between Challenge Issuance and Token Redemption.

4.4. Freshness and Reuse of the VDT

VDTs are intended to be a limited-use tokens which is guaranteed by the protocol through token validity window and nonce-based challenge seeds. More details on this are covered in section 8.

4.5. Privacy Properties

The protocol is designed with the following privacy properties.

* The Issuer is not able to determine when or where a token is redeemed.

* The Verifier is not able to determine when or why a token was issued.

* The Client does not convey the client identifier in Challenge Issuance and Token Redemption.

4.6. Error Handling

The failure responses in the protocol MUST NOT reveal whether the token was previously redeemed or is partially valid.

4.7. Extensibility

The protocol is designed to support extensions. Extensions MUST preserve the security and privacy properties defined in this document. Some of the envisaged future extensions are listed below.

* Multiple VDF algorithms.

* Multiple token encodings such as CBOR, JSON, binary TLV, compact byte strings.

* Multiple transport protocols such as HTTP, OHTTP.

5. VDF Requirements and Constructions

VDT protocol relies on the cryptographic primitive VDF. Although the protocol does not mandate a specific VDF construction, consistent with the approach taken in other cryptographic protocol specifications RFC 9180, it is expected that the construction MUST have certain properties. This section presents the VDF and their expected properties.

5.1. VDF

A VDF is a cryptographic primitive that requires a prescribed number of sequential steps for the computation even with massive parallel processing capability but the result can be verified efficiently. A VDF is defined by the following three associated algorithms.

* Setup: This algorithm takes security and difficulty parameters and produces public parameters consisting of an evaluation key and a verification key.

* Eval: This algorithm takes an input value and the evaluation key to produce an output value and a proof. This requires a sequential computation and the computation time is directly proportional to the difficulty parameter.

* Verify: This algorithm takes the input value, the proof, the output value and the verification key and efficiently produces a Boolean value indicating whether the output value matches the result of the Eval algorithm when given input value and the evaluation key.

VDF is used in the VDT protocol to cryptographically ensure minimum elapsed time between the Challenge Issuance and the Token Redemption events.

5.2. VDF Properties

A VDF used in the protocol MUST have the following properties. These properties are formalized in existing VDF constructions [BONEH-VDF], [WESOLOWSKI], [PIETRZAK].

1. Sequentiality: The computation of the VDF MUST require a number of sequential steps proportional to the delay parameter. It SHOULD NOT be possible for an adversary to be able to compute the result more efficiently through parallelism. This property enables the enforcement based on the elapsed time and not the computational resources.

2. Efficient Verification: The verification of the VDF output MUST be efficient. This property enables scalable verification.

3. Determinism and Uniqueness: Output of the VDF MUST be unique for a given input and a set of parameters. This property enables the deterministic verification.

4. Public Verifiability: The verification MUST be possible using only public information without the need to access any secret information maintained by the Issuer or the Client.

5. Parametrized Delay: The delay parameter of the VDF MUST be tunable to let the Issuer change the intended minimum elapsed time as required. The specific mapping of the delay parameter to the wall-clock time is deployments-specific and is kept out of scope of this document.

6. Security Assumptions: ONLY the VDF constructions which rely on well-studied underlying cryptographic hardness assumptions MUST be used and the constructions which use proprietary or undisclosed trapdoors are NOT RECOMMENDED.

5.3. VDF Constructions

This document does not endorse any specific VDF construction. Any VDF construction with properties mentioned in Section X.X MAY be used. The two constructions (there could be more) with these properties are Wesolowski-style VDFs and Pietrzak-style VDFs.

All cryptographic operations MUST use domain separation information to mitigate the risk of cross-protocol or cross-context attacks. This prevents the possibility of using a cryptographic value from one context in another context across multiple protocol instances just because both the instances share the same underlying primitive.

The inputs to the VDF MUST be created from the challenge parameters provided by the Issuer. A VDF input MUST include a challenge seed provided by the Issuer, context information (if used) and domain separation information.

```
vdf_input = Hash("VDT-VDF-Input" ||
                  challenge_seed ||
                  encode(delay_parameter) ||
                  context_binding
            )
```

The VDF output consists of a computed output value and an associated proof that allows for efficient verification. Both the output and the associated proof MUST be included in the token.

The protocol is intended to be generic in the use of VDF constructions and implementations SHOULD support the use of new VDF constructions without breaking interoperability. Tokens computed with unsupported or deprecated parameters MUST be rejected by Issuers and Verifiers.

6. Verifiable Delay Token

This section defines the VDT structure which consists of not only the result of a completed VDF computation but also the metadata to enable efficient and stateless verification. The structure is independent of both the transport and the encoding methods. A VDT binds the challenge seed, delay parameter, optional context provided by an Issuer with a VDF computation and proof provided by a Client. A token MUST NOT contain any type of client identifier, network address

or value that can be used to establish a connection across Challenge Issuance and Token Redemption.

6.1. Token Structure

A VDT consists of the following fields.

1. token_version: An identifier indicating the token format. This field MUST be present. Verifiers MUST reject tokens that have unsupported versions.

2. issuer_id: An identifier indicating the Issuer which generated the challenge included in the token. This field MUST be present. This MUST be verifier-recognizable value and MAY be publicly available.

3. challenge_seed: A value provided by the Issuer and used in the VDF computation by Client. This field MUST be present. This MUST be highly random, SHOULD NOT be predictable and MUST NOT be reused across challenges.

4. delay_parameter: A value provided by the Issuer and used in the VDF computation by Client such that the computation time is proportional to this parameter. This field MUST be present. Verifier MUST validate that the value is acceptable for the intended use.

5. vdf_output: The value produced by VDF computation by Client and used by Verifier to validate the token. This field MUST be present. This output value MUST be uniquely computed from a challenge seed, a delay parameter and an optional context.

6. vdf_proof: A cryptographic proof produced by VDF computation by Client and used by Verifier to validate the token. This field MUST be present. This proof MUST be uniquely computed from a challenge seed, a delay parameter and an optional context.

7. context_binding (optional): A value provided by the Issuer which is used in the VDF computation. This field MAY be present. If present in the token, the value MUST be used in VDF computation. If the intended use requires context binding, the verifier MUST reject tokens missing this field.

8. validity_window (optional): A value indicating expire time beyond which the token is deemed invalid. This field MAY be present. If present in the token, it MUST NOT require synchronized clocks between the Issuer and the Verifier. This field SHOULD be used only to limit the token usage within a certain time.

```
VerifiableDelayToken {
    uint8    token_version;
    opaque   issuer_id;
    opaque   challenge_seed;
    uint64   delay_parameter;
    opaque   vdf_output;
    opaque   vdf_proof;
    opaque   context_binding;     // OPTIONAL
    opaque   validity_window;     // OPTIONAL
}
```

6.2. Token Construction

The VDT is created by a Client. The Client computes a VDF with the inputs of a challenge seed, a delay parameter, an optional context and a protocol-specific domain separation string. The VDF output value, VDF proof and the metadata are embedded into the token.

6.3. Token Verification

The Verifier verifies the VDT by validating the following.

1. Token version is supported.

2. Issuer identifier is recognized and trusted.

3. Challenge seed and delay parameter.

4. Context against the expected use.

5. VDF proof verifies correctly against the output.

6. The token is not expired and is not used earlier.

The Verifier MUST reject the token if any of these checks fail.

6.4. Privacy Properties of the Token Format

The token format ensures privacy by the following facts.

* No inclusion of Client identifiers.

* No inclusion of timestamps.

* No inclusion of network identifiers.

* Unlinkability across Challenge Issuance and Token Redemption across protocol instances.

Specific implementations of the protocol MUST NOT include any additional fields that compromise these properties.

6.5. Token Encoding

The token format is generic in the use of token encoding. The encodings MUST be able to encode individual fields with clear boundaries. Implementations MAY use CBOR, Binary HTTP or other compact binary encoding formats. However, since text-based encodings such as JSON have limitations related to size and parsing, they are NOT RECOMMENDED.

6.6. Extensibility

The token format is extensible. The future specifications MAY define new fields. If the token contains an unknown optional field, it MUST be ignored by default unless explicitly required. Semantics of the fields MUST NOT be changed by new token versions.

7. Protocol Messages (Wire-Level Flows)

This document defines the VDT protocol using following messages.

Message	Direction	Purpose
Challenge Request	Client -> Issuer	Request challenge seed and delay parameter
Challenge Response	Issuer -> Client	Provide VDF challenge
Token Redemption Request	Client -> Verifier	Provide VDT
Verification Response	Verifier -> Client	Verifier’s decision on acceptance or rejection

Table 1: Protocol Messages

7.1. Challenge Request

The Challenge Request message is sent by the Client to the Issuer to request parameters required for delay computation.

This message MAY include the following parameters.

* supported_vdfs: A list of VDF constructions supported by the Client.

* delay_hint (Optional): A value indicating the desired delay range. Issuers MAY ignore this parameter.

* context_request (Optional): A Boolean value indicating a request to include context the Challenge Response message.

Any information which can be used to identify the client such as client identifier MUST NOT be included in this message.

Issuer MUST treat Challenge Request message as unauthenticated and MAY apply policy such as rate-limitation.

7.2. Challenge Response

The Challenge Response message is sent by the Issuer to the Client to provide the parameters required for delay computation. This message is sent in response to an earlier received Challenge Request message.

This message MUST include the following parameters.

* Issuer_id: A value indicating the Issuer.

* challenge_seed: A fresh, unpredictable value.

* delay_parameter: A value indicating the required delay for the VDF.

* vdf_parameters: A set of values required for the VDF computation.

This message MAY optionally include the following additional parameters.

* context_binding: A value related to the context which is to be included in the VDF input.

* validity_window: A time indicating the expiration time of the token.

Issuer MUST not include any information which can either identify the client or can be used to correlate between the Challenge Request and the Challenge Response messages, MUST ensure that the challenge_seed is never reused and MUST ensure that the parameters provided are consistent with the selected VDF construction.

7.3. Delay Computation (by Client)

After receiving the Challenge Response message, the Client computes the VDF using challenge_seed, delay_parameter and optional context_binding to generate vdf_output and vdf_proof. The Client constructs a VDT with the structure mentioned in Section 7. The client MAY be offline during the computation.

7.4. Token Redemption Request

The Token Redemption Request message is sent by the Client to the Verifier to prove that a certain duration of the time has elapsed since the challenge was issued.

This message MUST include the following parameters.

* verifiable_delay_token: A VDT.

This message MAY optionally include the following additional parameters.

* requested_resource: A value indicating the resource to be accessed.

* context_confirmation: A Boolean value indicating a request to bind context.

Verifier MUST treat Token Redemption Request message as unauthenticated, MUST NOT require the Client to include information which can be used to identify clients and MUST perform the validation steps defined in Section 8.

7.5. Verification Response

The Verification Response message is sent by the Verifier to the Client to provide the verification result. This message is sent in response to an earlier received Token Redemption Request message.

This message MUST include the following parameters.

* result: A Boolean value indicating whether the token is valid considering the delay requirement in the token.

Verifier MUST NOT reveal whether the token was previously redeemed, MUST NOT reveal whether the token is partially valid and SHOULD minimize the response size.

7.6. Replay Protection

The protocol prevents against replay attacks by the following mechanisms.

* Token uniqueness via challenge_seed

* Token usage period via validity_window

* Replay detection via Verifier checks.

7.7. Transport Independence

The protocol messages are agnostic to the underlying transport mechanism. Messages MAY be conveyed over HTTP, OHTTP or other secure transports. The underlying transport mechanism MUST preserve message integrity.

7.8. Error Handling

The protocol messages can error out because of reasons such as unsupported VDF construction, invalid parameters, expired or reused token. Implementations MUST consider all errors as critical.

8. Privacy Considerations

This section presents the privacy properties of the protocol. The protocol aims to enable time-based access control without leaking client identification, cross-request linkability and unnecessary disclosure of information. The privacy analysis in this section follows the guidance in RFC 6973. Protection against traffic analysis is out of scope of the protocol.

* Absence of Client Identifiers

The protocol does not require Clients to include client identity, network address, device identity or issuance timestamp either in Challenge Request message or Token Redemption Request message. The protocol also does not require authentication or session establishment of Client with either the Issuer or the Verifier which prevents Issuer to correlate issued challenge seed with the Client and also prevent Verifier to correlate received token with the Client.

* Unlinkability Between Issuance and Redemption

The protocol does not require the Issuer to learn about either the delay computation or the token redemption and the Verifier to learn about the challenge issuance. The token contains only the challenge seed which uniquely identify the issuance transaction but the challenge seed does not reveal client identity. Assuming that the Issuer and the Verifier do not collude, the Challenge Request message and the Token Redemption Request message cannot be linked.

* Minimal Information Disclosure

The protocol messages reveal only the strictly necessary required information. Challenge Response message conveys the delay parameter, the VDF parameters and the issuer metadata required for verification. Token Redemption Request message conveys the VDF computed output value and a proof. None of the messages include issuance or redemption timestamps. This avoids revealing behavioral patterns of Client.

* Resistance to Cross-Token Correlation

The protocol ensures the resistance to any correlation among VDTs by ensuring that no two VDTs can be derived from the same challenge seed and each challenge seed generates a cryptographically unique VDT. Implementations MUST avoid introducing additional information that could enable such correlation.

* Context Binding and Privacy

The protocol introduced the usage of optional context to allow a Verifier to restrict token usage to a specific scope such as for a particular resource or a particular service. Though the use of context binding does not aim to introduce client identifier, overly specific or Client derived context values could reduce Client anonymity. Verifiers SHOULD ensure that context values are appropriate and does not reveal Client identity.

* Use of Relays and Oblivious Transports

The protocol can be deployed over privacy-preserving transports such as OHTTP that hide Client network metadata. These transports ensure that neither Issuers nor Verifiers learn Client network address and Relays cannot access cryptographic content. Deployments that do not use such privacy-preserving transports may expose network addressees and have to take appropriate measures to prevent revealing Client identity.

* Token Reuse and Privacy Trade-offs

Verifiers may track token usage to prevent against replay attacks. Verifiers SHOULD take appropriate measures so that replay detection does not introduce unintended linkability, SHOULD avoid maintaining a per-client state and SHOULD require minimum necessary information such as reuse of challenge seed within a limited window.

* Comparison with Alternative Mechanisms

The alternative time-based or abuse mitigation mechanisms such as keeping server-side timers, proof-of-work or CAPTCHA systems have privacy limitations. Server-side timers require client identification and enable linkability. Proof-of-work is proportional to computational power and the not the sequential computation and also enables fingerprinting. CAPTCHA systems reveal interaction patterns and tracking. The VDT protocol provides time enforcement while avoiding these privacy limitations.

* Residual Privacy Risks

Some of the privacy risks are deliberately kept out of scope as they are specific to the deployment scenarios. For example, without the use of privacy-preserving transports, network addresses may be observed which may leak client identify. An adversary may observe redemption patterns and perform side-channel based timing attacks. An adversarial Issuer and an adversarial Verifier may collude to compromise unlinkabilty. Deployments SHOULD take appropriate measures to avoid such privacy risks.

9. Security Considerations

This section presents the security threats in the protocol and the mechanism by which they are mitigated.

* Correctness of Delay Enforcement

The protocol assumes that the selected VDF construction enforces sequential computation and does not permit the computation to be done significantly faster by means such as shortcut evaluation or parallel speedup. This assumption is critical to the protocol because otherwise an adversary may compute tokens faster than the intended delay. ONLY those VDF constructions which are well-studied and publicly documented should be used. The protocol does not aim to compensate performance difference in different hardware. Since different hardware can have different performance, the protocol provides only approximate delay guarantees which indicate relative and not absolute wall-clock time.

* Token Forgery

An adversary may attempt to generate a valid VDT without performing the required delay computation. The protocol mitigates the token forgery attacks by cryptographic security of the VDF which ensures that forging a token by avoiding or reducing the required delay computation is computationally infeasible. Verifier MUST reject tokens that fail verification. If at any time, a VDF construction is found vulnerable to forgery, tokens derived from that construction MUST be considered insecure.

* Replay Attacks

VDTs are bearer token and an adversary may replay it while it is still valid. This behavior is inherent to bearer tokens and does not reduce the security of delay enforcement. The protocol mitigates the replay attacks by preventing token reuse through unique challenge_seed, restricting usage time period through validity_window and binding token to a specific context (resource, operation or request) through context_binding.

* Man-In-The-Middle Attacks

An adversary may observe, modify, replay or inject protocol messages as they are communicated among Client, Issuer and Verifier. The security of the protocol relies on the cryptographic validity of the token in the message rather than the confidentiality of the message. Any modification in the protocol messages makes the token invalid which results in failed verification. Any forging of a VDT in a message without performing the required delay computation is computationally infeasible. Any leakage of protocol messages does not reveal client identifiers. An adversary may tamper the challenge in the Challenge Response message which makes the Client compute VDF over adversarial challenge which later will fail verification in Verification Response message. This may result in Denial-of-Service which is not a security breach and is inherent to bearer tokens. To prevent this, Issuer SHOULD authenticate the challenge in the Challenge Response message. But, even if an adversary tampers the challenge, it cannot create a valid token, learn client identifier or impersonate the Client. It is RECOMMENDED to use secure transport mechanism such as TLS or OHTTP to reduce passive observation but it is not required for cryptographic correctness.

* Token Rebinding and Context Confusion

To prevent the usage of a token in a context different from the intended one, the token MAY be bound to an application-defined context. If present, the context value MUST be used in VDF computation. Verifiers MUST reject tokens whose context does not match the expected value. Context values SHOULD be canonicalized prior to inclusion.

* Issuer Misbehavior

An adversarial Issuer may issue reused challenge seeds, weak parameters or insufficient delay requirements. The protocol mitigates this by ensuring that Verifiers MUST validate Issuer identifiers to find whether the Issuer is trusted and SHOULD reject tokens which do not adhere to expected parameter constraints. Verifiers SHOULD validate that delay parameters meet locally configured minimum security thresholds.

* Verifier Misbehavior

An adversarial Verifier may attempt to extract additional information by requesting additional unnecessary information. Clients SHOULD treat Verifiers as potentially untrusted entity, SHOULD avoid revealing any additional information beyond what is mentioned in this document and MAY decline to provide additional information not required for verification. The protocol does not prevent an adversarial Verifier from applying arbitrary access policies after token verification.

* Denial-of-Service Considerations

Issuers may receive large number of Challenge Request messages and Verifiers may receive large number of invalid or malformed Token Redemption Request messages. This may result in denial-of-service attacks. Implementations SHOULD use standard DoS mitigation techniques such as rate limiting, request validation and bounded resource allocation. Verifier SHOULD verify the token only after preliminary validation of token structure is done.

* Parameter Selection Risks

Improper selection of delay parameters can have an impact on the security. A too small value may fail to prevent attacks and a too large value may cause excessive consumption of Client resources. This document does not specify parameter values as it largely depends on the intended use case. Deployments MUST select parameters suitable for the intended use case.

* Side-Channel Considerations

The protocol does not explicitly address the side-channel attacks that may arise from means such as timing differences in the verification process, observable network behavior, computation patterns at the Client. Deployments SHOULD take appropriate measures to prevent such attacks.

* Collusion Between Issuer and Verifier

An adversarial Issuer and an adversarial Verifier may collude to correlate the Challenge Issuance Request message and Token Redemption Request message. This compromises the unlinkability of Client across these two messages. This protocol assumes non-collusion between Issuer and Verifier. Deployments SHOULD take appropriate measures to prevent such collusions.

* Cryptographic Agility and Deprecation

It is possible that some of the secure VDF constructions may become insecure in future because of change in cryptographic assumptions. Deployments MUST support mechanisms to deprecate such insecure VDF constructions and support to use the updated parameters. Verifiers SHOULD reject tokens generated from deprecated parameters.

10. Deployment Considerations

This section discusses the practical considerations of implementing the VDT protocol. These considerations do not affect protocol requirements and are informative.

* Parameter Selection

Choice of delay parameter directly impacts the security and resource consumption. Shortening this value may reduce the strength to prevent attacks and increasing this value has a direct effect on consumption of Client resources. Deployments SHOULD select a balanced value which balances these two aspects. This document does not recommend a specific value since it largely depends on the operational constraints and Client capabilities.

* Client Resource Constraints

The protocol requires Clients to perform sequential computation for the duration of the delay. Deployments SHOULD consider the Client device resources such as battery consumption and thermal constraints for selecting the suitable delay. Clients MAY abort the delay computation and Issuers SHOULD avoid issuing challenges which may result in unreasonable long delay computations for a typical Client device.

* Issuer Load and Scalability

Issuers are primarily involved in generating delay challenges and are not required to maintain persistent per-client state. Hence, Issuers can use suitable mechanisms to scale horizontally. This document does not recommend a specific such mechanism as it largely depends on the particular deployment. Issuers MAY apply standard request filtering and rate limiting to avoid denial-of-service attack.

* Verifier Load and Verification Cost

Verifiers are primarily involved in verifying the received VDTs. The verification process is assumed to be efficient. To prevent against denial-of-service attacks, Verifiers SHOULD apply inexpensive structural validation prior to cryptographic verification.

* Replay Detection State

Verifiers may need to maintain state such as recently seen challenge seed or token identifier to protect against replay attacks. Deployments SHOULD minimize the duration for which such state is maintained to avoid introducing the linkability. Stateless verification can be achieved to prevent replay attacks by enforcing short token validity windows.

* Trust and Governance of Issuers

When Issuers and Verifiers are different entities, Verifiers must decide whether the issuer is a trusted entity. This can be done through static out-of-band agreements and public key distribution.

* Incremental Deployment

The protocol may be used along with existing alternative mechanisms. For example, a service MAY offer CAPTCHA to some clients and VDT to other clients, MAY use both CAPTCHA and VDT to prevent against rate-limiting and MAY require VDTS only in certain cases such as repeated authentication failures.

* Deployment with Oblivious Transports

Using privacy-preserving transports such as OHTTP can provide stronger privacy guarantees by hiding client network metadata. However, they may introduce additional latency and infrastructure requirements. Deployments that require such stronger privacy guarantees SHOULD consider the associated overheads.

* Failure Handling and User Experience

The protocol may generate error responses due to several reasons such as invalid parameters, expired tokens or verification errors. Verifiers SHOULD ensure that only minimal necessary information is provided without any information which can leak client identity.

* Monitoring and Abuse Evaluation

Any observatory or analytical system such as system monitoring or the abuse mitigation effectiveness SHOULD be performed in a privacy-preserved manner and MUST NOT either rely or leak any information which can reveal client identity.

11. Composition with Existing Protocols (HTTP, OHTTP, Privacy Pass)

This section describes how VDT protocol may be used with existing Internet protocols.

11.1. Composition with HTTP

VDT protocol is designed to work naturally with HTTP-based systems. Before processing an HTTP reqest, a Verifier MAY require the Client to present a VDT. The Client MAY include the token in HTTP request header or in a request body. For example, a Client MAY include the token in an authorization header.

Alternatively, a Client MAY include the token in HTTP request body when using non-idempotent methods. Specific HTTP header names or status code are left to the future specifications or application profiles.

11.2. Stateless Access Control in HTTP Services

The protocol allows HTTP services to enforce time-based access control without requiring to maintain per-client state. When a service receives an HTTP request containing a VDT, it verifies the token (as mentioned in Section X.X), takes a decision based on verification outcome and discards the token along with an associated state. This lets HTTP services enforce delays without tracking Clients across requests.

11.3. Composition with OHTTP

OHTTP and its properties are defined in [RFC9458]. The VDT protocol works well with OHTTP as well. Challenge Request, Challenge Response, Token Redemption Request and Verification Response MAY be encapsulated in OHTTP messages and conveyed via an OHTTP Relay. OHTTP prevents Issuers and Verifiers from learning client or network identifiers. OHTTP is OPTIONAL but RECOMMENDED where network-layer

privacy is required. Figure 4 illustrates composition with OHTTP.

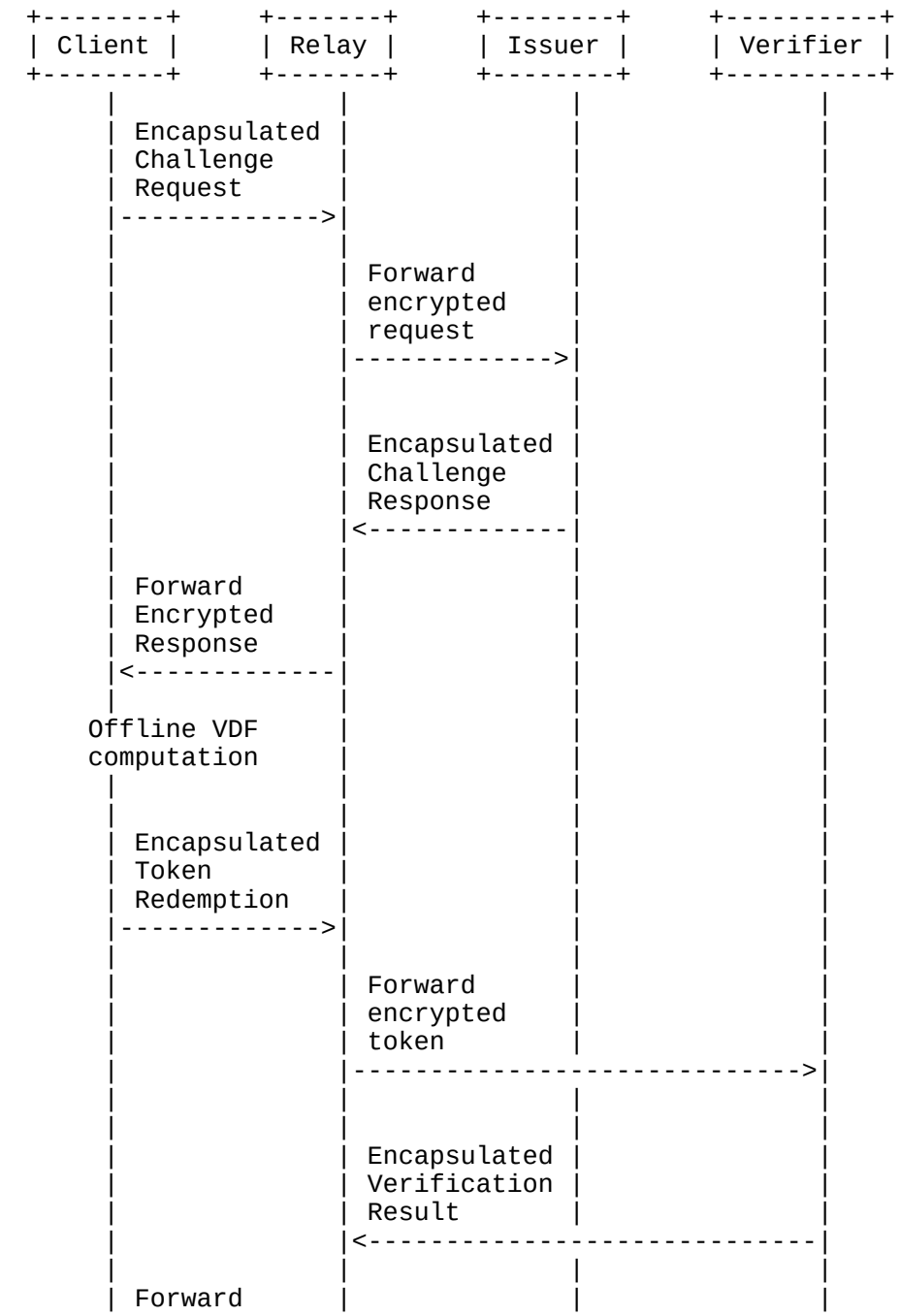

Figure 4: Composition with OHTTP

11.4. Composition with Privacy Pass

The VDT protocol is complementary to Privacy Pass. Privacy Pass can be used to create anonymous and unlinkable tokens that prove that a client has passed a challenge such as a CAPTCHA or a device attestation. VDT protocol can be used to create tokens that a minimum amount of time has elapsed. VDTs MAY be used as a time-delay requirement for the issuance of a Privacy Pass token. Conversely, a privacy pass token MAY be used for the challenge issuance. Deployments which use VDT protocol along with Privacy Pass MAY enable different scenarios. A service may implement an anonymous time-based abuse mitigation by requiring a Client to present a VDT after first attempt without disclosing Client identity. A service may implement a rate-limit anonymous API access by requiring a Client to present a VDT through OHTTP. A service may implement privacy-preserving delays by requiring a Client to present a VDT instead of interactive CAPTCHA or solving puzzles. Figure 5 illustrates composition with Privacy Pass.

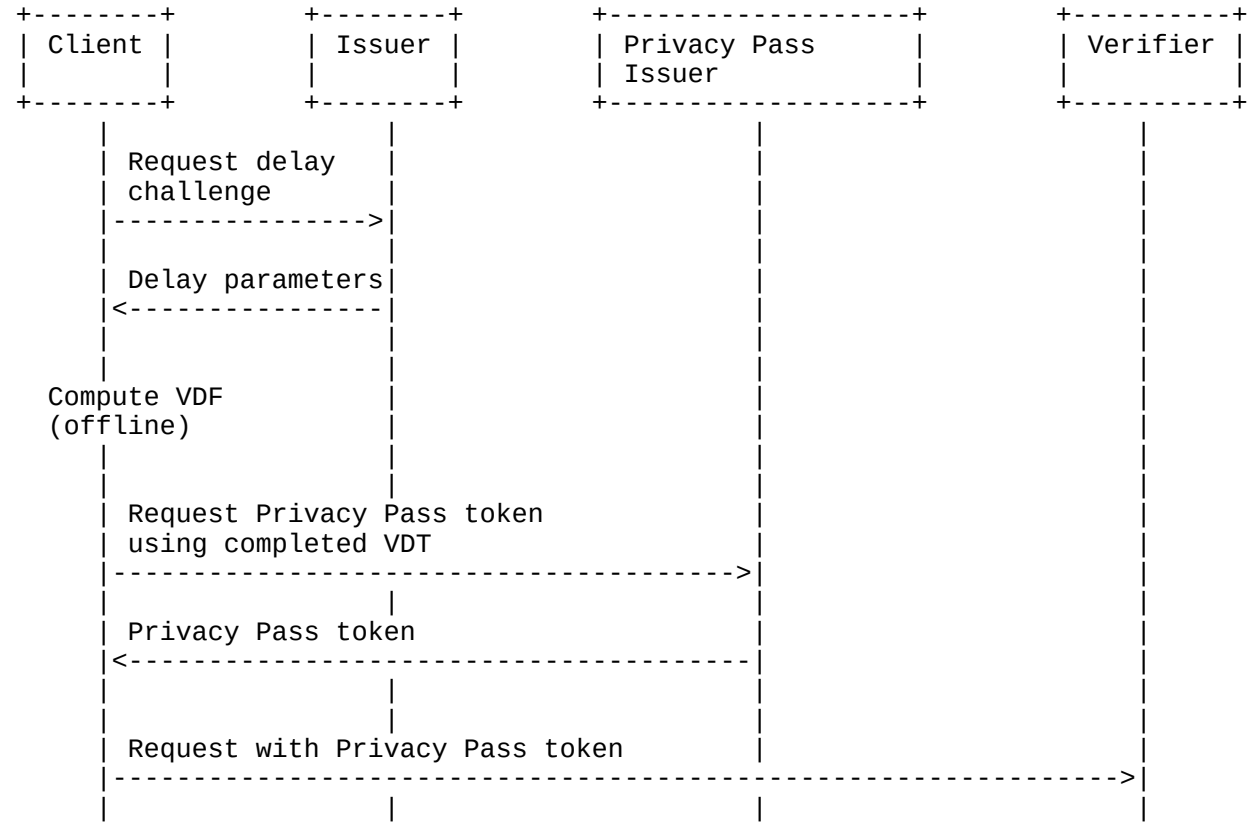

Figure 5: Composition with Privacy Pass

11.5. Interaction with Transport Security

The VDT protocol assumes that the underlying transport provides necessary integrity and confidentiality to the messages. HTTP-based transports SHOULD use TLS. When network-layer privacy is required, OHTTP SHOULD be used.

12. IANA Considerations

This document does not require any immediate IANA actions. However, future extensions MAY require some optional IANA actions.

12.1. Token Type Identifier Registry (Optional)

It is possible that in future, similar to VDTs, other type of tokens is proposed and defined for different use cases. Future specifications MAY define a token type identifier registry to identify the tokens. If such a registry is created, it is RECOMMENDED that it includes a token type identifier, a brief description of the token and a reference to the defining specification. The registration policy for such a registry SHOULD be _Specification Required_.

12.2. HTTP Header Fields (Optional)

This document does not introduce any new HTTP header fields. Future specifications MAY introduce a new HTTP header field. Any future specifications that introduce a new HTTP header field for transmitting VDTs MUST be registered with IANA. This document does not request such registration.

12.3. Media Types (Optional)

This document does not introduce any new media types. Future specifications MAY introduce new media types for encapsulating VDTs or protocol messages. Any future specifications that introduce a new such media type MUST be registered with IANA. This document does not request such registration.

12.4. Cryptographic Algorithm Registries

This document does not introduce any modifications to cryptographic algorithm registries. Cryptographic agility is achieved through protocol negotiation and specific deployments rather than through central registry.

13. Implementation Status

This section presents the implementation status of the VDT protocol.

* Note to RFC Editor:

This section SHOULD be removed before publication. as an RFC. This section describes the present known implementations of the VDT protocol according to the RFC 7942 with the aim to present the interoperability feasibility.

* Reference Implementation

At the time of writing this document, a reference implementation of the VDT protocol is under development. The reference implementation is written in Python as a standalone library without HTTP or OHTTP integration to demonstrate a proof-of-concept protocol level implementation. The scope is limited to the challenge generation, VDF computation, token construction, token verification and protocol messages. Wesolowski VDF construction is used at the experimental level. The primary goal of the prototype is to validate protocol feasibility and parameter selection and not to optimize for performance or production use.

* Experimental Integration

An experimental integration is planned with other Internet protocols such as HTTP-based deployment using a custom header, OHTTP encapsulation for challenge issuance and token redemption and Privacy Pass with delay challenges instead of interactive challenges. These integrations are planned only to test the interoperability.

* Security Considerations for Implementations

No formal review of the prototype is done so far. Parameters are selected only for experimental purpose and no evaluation against attacks has been done. The prototype SHOULD NOT be used in production environments.

* Future Implementations

Implementations in programming languages other than Python are expected to be developed to support different deployments. Independent implementations are encouraged and feedback from the implementations are welcome.

14. Limitations

This section describes limitations of the VDT protocol. These limitations are intrinsic to the problem and cryptographic primitives and should not be considered as flaws in the protocol design.

* Approximate Time Enforcement:

The protocol only enforces a minimum amount of sequential computation which is directly related to the elapsed time and is not an indication of the exact wall-clock duration. High performant hardware devices may perform delay computation faster than the less performant ones. The protocol does not guarantee fairness across heterogeneous hardware devices and hence provides a relative time enforcement rather than an absolute wall-clock guarantee.

* No Absolute Time or Scheduling Guarantees:

The protocol does not provide guarantees about absolute timestamps, timestamps synchronized with real-world clocks or about any time completion beyond the minimum required delay requested. Deployments which require exact or synchronized timestamps MUST rely on additional mechanisms which are outside the scope of this document.

* Resource Consumption on Clients:

The delay computation on Client may have an effect on battery life, thermal limits or responsiveness on constrained devices. Clients may abort computation at any time which results in unusable token. The protocol does not provide device-aware parameter negotiation.

* No Identity, Accountability, or Attribution:

One of the core goals of the protocol is not to reveal Client identity. Hence, Verifiers cannot correlate tokens to Clients, per-client personalized policies cannot be applied and abuse mitigation is limited to time-based controls. If a deployment needs an identity-based control, additional mechanisms MUST be used.

* Limited Protection Against Sophisticated Adversaries:

The protocol does not aim to prevent against adversaries with specialized hardware that significantly accelerates sequential computation, side-channel attacks on Client computation, traffic analysis at network level and collusion between Issuer and Verifier. Mitigation against these threats are out of scope.

* Dependence on VDF Maturity:

The security of the protocol depends on the properties of the selected VDF construction. Future advances in cryptanalysis may weaken existing VDF constructions and performance may vary across constructions. Though the protocol does not fully eliminate this risk, it mitigates the risks through cryptographic agility.

* No Abuse Detection or Policy Definition:

The protocol explicitly does not define what constitutes an abuse, when a delay should be required and what actions should be taken for token acceptance or rejection. These decisions are left to the specific deployments.

* No Economic or Incentive Guarantees:

VDTs are not a payment mechanism and any economic interpretation of VDTs is out of scope of this document. The protocol neither defines an economic cost nor a pricing model.

15. Future Work

This section presents some of the possible future works of this document. These works are out of scope for this document and do not affect the VDT protocol defined herein.

* VDF Profiles:

This protocol has limited itself to the required VDF properties and is kept generic to various VDF constructions. One possible future work could be to define VDF profiles for a specific set of VDF constructions, recommended parameter ranges or performance characteristics.

* Performance Evaluation and Benchmarking:

Another future work could be performance evaluation and benchmarking across heterogeneous hardware of various metrics such as energy consumption on constrained devices and verification costs. Deployment environments can be benefitted from benchmarks in selecting the suitable parameters.

* Adaptive Delay Negotiation:

This protocol does not support adaptive negotiation which can issue parameters such as delay_challenge based on Client’s capabilities such as hardware device. Future works may introduce such negotiation with the caution to maintain privacy of Client identifiers.

* Integration Profiles:

Though this document describes integration with HTTP, OHTTP and Privacy Pass, it does not define normative bindings. Possible future work could define normative bindings such as standard HTTP header fields or media types and profiles for Privacy Pass integration.

* Formal Security Analysis:

Another future work could be to do the formal security analysis of the protocol.

* Broader Applications:

Though this document primarily focuses on time-based access control, the protocol can be extended to other broader applications such as delayed revelation and staged access to a resource.

16. References

16.1. Normative References

[RFC9458] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/info/rfc9458>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

16.2. Informative References

[BONEH-VDF] Boneh, D., "Verifiable Delay Functions", 2018.

[WESOLOWSKI] Wesolowski, B., "Efficient Verifiable Delay Functions", 2018.

[PIETRZAK] Pietrzak, K., "Simple Verifiable Delay Functions", 2018.

Author’s Address

Puneet Bakshi
C-DAC
Pune
Email: puneetb@cdac.in