

**COMPUTER-IMPLEMENTED METHOD, COMPUTER PROGRAM AND
COMPUTER-READABLE MEDIUM USING A CHAMELEON HASH FUNCTION**

TECHNICAL FIELD

- 5 **[0001]** Various aspects relate to a computer-implemented method, a computer program and a computer-readable medium using a chameleon hash function.

BACKGROUND

- 10 **[0002]** In public-key cryptography, digital signatures are used to ensure integrity and authenticity of data. A signer having the knowledge of a certain secret key (or signing key) can sign a message such that anyone with the corresponding public key (or verification key) can check the validity of the signature. A digital signature scheme Σ typically consists of three polynomial-time algorithms. The key generation algorithm KeyGen takes a security parameter λ as input, and it outputs a secret key sk and the corresponding public key pk . The signing algorithm Sign takes sk and a message $m \in M$ (for a message space M) as input, and it outputs a signature σ . The verification algorithm Verify takes pk , the message m and the signature σ as input, and it outputs accept (or reject) if the signature is valid (or invalid). Due to the unforgeability property of a secure digital signature scheme, it is not possible to sign a message on someone else's behalf without having the corresponding secret key, unless
- 20 the signature for that message is already available.

- [0003]** A need exists to provide an improved method for signing a message and verifying the signed signature for the message.

SUMMARY

[0004] According to a first aspect of the present disclosure, a computer-implemented method using a chameleon hash function is provided. The chameleon hash function may include one or more modular exponentiations and one or more modular multiplications, the method including, at a signature-verification phase: precomputing, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function; and evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations and one or more modular multiplications.

10 [0005] According to a second aspect of the present disclosure, a computer program including instructions to execute the steps of the computer-implemented method as described herein is provided.

[0006] According to a third aspect of the present disclosure, a non-transitory computer-readable medium having stored thereon the computer program as described herein is provided.

15 [0007] According to a fourth aspect of the present disclosure, a system using a chameleon hash function is provided. The chameleon hash function may include one or more modular exponentiations and one or more modular multiplications, the system including: a processor configured to, at a signature-verification phase: precompute, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function; and evaluate, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations.

[0008] Throughout the drawings, it should be noted that like reference numbers are used to depict the same or similar elements, features, and structures. The drawings are not necessarily to scale, emphasis instead generally being placed upon illustrating aspects of the disclosure. In the following description, some aspects of the disclosure are described with
5 reference to the following drawings, in which:

FIG. 1. is a flowchart showing an example computer-implemented method using chameleon hash function, according to various embodiments of the present disclosure;
FIG. 2 is a flowchart showing an example offline phase of the computer-implemented method using chameleon hash function, according to various embodiments of the
10 present disclosure;

FIG. 3 is a flowchart showing an example online phase of the computer-implemented method using chameleon hash function, according to various embodiments of the present disclosure; and

FIG. 4 is a block diagram showing an example electronic device, according to various
15 embodiments of the present disclosure.

DETAILED DESCRIPTION

[0009] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and aspects in which the disclosure may be
20 practiced. One or more aspects are described in sufficient detail to enable those skilled in the art to practice the disclosure. Other aspects may be utilized and structural, logical, and/or electrical changes may be made without departing from the scope of the disclosure. The various aspects of the disclosure are not necessarily mutually exclusive, as some aspects can be combined with one or more other aspects to form new aspects or embodiments. Various

aspects are described in connection with methods and various aspects are described in connection with devices. However, it may be understood that aspects described in connection with methods may similarly apply to the devices, and vice versa.

[0010] It should be understood that the singular terms "a", "an", and "the" include plural references unless context clearly indicates otherwise. Similarly, the word "or" is intended to include "and" unless the context clearly indicates otherwise.

[0011] It will be further understood that the terms "comprise" (and any form of comprise, such as "comprises" and "comprising"), "have" (and any form of have, such as "has" and "having"), "include" (and any form of include, such as "includes" and "including"), and "contain" (and any form of contain, such as "contains" and "containing") are open-ended linking verbs. As a result, a method or device that "comprises," "has," "includes" or "contains" one or more steps or elements possesses those one or more steps or elements, but is not limited to possessing only those one or more steps or elements. Likewise, a step of a method or an element of a device that "comprises," "has," "includes" or "contains" one or more features possesses those one or more features, but is not limited to possessing only those one or more features. Furthermore, a device or structure that is configured in a certain way is configured in at least that way, but may also be configured in ways that are not listed.

[0012] Approximating language, as used herein throughout the specification and claims, may be applied to modify any quantitative representation that could permissibly vary without resulting in a change in the basic function to which it is related. Accordingly, a value modified by a term or terms, such as "about," "substantially", is not limited to the precise value specified but within tolerances that are acceptable for operation of the embodiment for an application for which it is intended. In some instances, the

approximating language may correspond to the precision of an instrument for measuring the value.

[0013] As used herein, the phrase of the form of “at least one of A or B” may include A or B or both A and B. Correspondingly, the phrase of the form of “at least one of A or B or C”, or including further listed items, may include any and all combinations of one or more of the associated listed items.

[0014] The term “exemplary” may be used herein to mean “serving as an example, instance, or illustration”. Any aspect or design described herein as “exemplary” is not necessarily to be construed as preferred or advantageous over other aspects or designs.

10 [0015] The terms “at least one” and “one or more” may be understood to include a numerical quantity greater than or equal to one (e.g., one, two, three, four, [...], etc.). The term “a plurality” may be understood to include a numerical quantity greater than or equal to two (e.g., two, three, four, five, [...], etc.). The phrase “at least one of” with regard to a group of elements may be used herein to mean at least one element from the group consisting
15 of the elements. For example, the phrase “at least one of” with regard to a group of elements may be used herein to mean a selection of: one of the listed elements, a plurality of one of the listed elements, a plurality of individual listed elements, or a plurality of a multiple of listed elements.

[0016] The words “plural” and “multiple” in the description and in the claims expressly
20 refer to a quantity greater than one. Accordingly, any phrases explicitly invoking the aforementioned words (e.g., “a plurality of (objects)”, “multiple (objects)”) referring to a quantity of objects expressly refer to more than one of the said objects. The terms “group (of)”, “set (of)”, “collection (of)”, “series (of)”, “sequence (of)”, “grouping (of)”, etc., and

the like in the description and in the claims, if any, refer to a quantity equal to or greater than one, i.e. one or more.

[0017] The terms “processor” or “controller” as, for example, used herein may be understood as any kind of entity that allows handling data. The data may be handled according to one or more specific functions executed by the processor or controller. Further, a processor or controller as used herein may be understood as any kind of circuit, e.g., any kind of analog or digital circuit. A processor or a controller may thus be or include an analog circuit, digital circuit, mixed-signal circuit, logic circuit, processor, microprocessor, Central Processing Unit (CPU), Graphics Processing Unit (GPU), Digital Signal Processor (DSP), Field Programmable Gate Array (FPGA), integrated circuit, Application Specific Integrated Circuit (ASIC), etc., or any combination thereof. Any other kind of implementation of the respective functions, which will be described below in further detail, may also be understood as a processor, controller, or logic circuit. It is understood that any two (or more) of the processors, controllers, or logic circuits detailed herein may be realized as a single entity with equivalent functionality or the like, and conversely that any single processor, controller, or logic circuit detailed herein may be realized as two (or more) separate entities with equivalent functionality or the like.

[0018] The term “memory” detailed herein may be understood to include any suitable type of memory or memory device, e.g., a hard disk drive (HDD), a solid-state drive (SSD), a flash memory, etc.

[0019] The term “module” detailed herein refers to, or forms part of, or includes an Application Specific Integrated Circuit (ASIC); an electronic circuit; a combinational logic circuit; a field programmable gate array (FPGA); a processor (shared, dedicated, or group) that executes code; other suitable hardware components that provide the described

functionality; or a combination of some or all of the above, such as in a system-on-chip. The term module may include memory (shared, dedicated, or group) that stores code executed by the processor.

[0020] Differences between software and hardware implemented data handling may blur. A processor, controller, and/or circuit detailed herein may be implemented in software, hardware, and/or as a hybrid implementation including software and hardware.

[0021] The term “system” (e.g., a crypto system, a computing system, etc.) detailed herein may be understood as a set of interacting elements, wherein the elements can be, by way of example and not of limitation, one or more mechanical components, one or more electrical components, one or more instructions (e.g., encoded in storage media), and/or one or more processors, and the like.

[0022] The term “first”, “second”, “third” detailed herein are used to distinguish one element from another similar element and may not necessarily denote order or relative importance, unless otherwise stated. For example, a first transaction data, a second transaction data may be used to distinguish two transactions based on two different foreign currency exchange.

[0023] The term “message” as used herein may be understood to include information in any suitable analog or digital form communicated among parties, e.g., provided as a file, a portion of a file, a set of files, a signal or stream, a portion of a signal or stream, a set of signals or streams, and the like. Further, the term “message” may also be used to mean a reference to information, e.g., in form of a pointer. The term “message”, however, is not limited to the aforementioned examples and may take various forms and represent any information as understood in the art. Any type of information, as described herein, may be

handled for example via one or more processors in a suitable way, e.g., as message. The term “message” may be interchangeably used with “data”. In cryptography, messages may be signed by a secret key or a private key by a signer and verified by a verifier using a matching or associated public key or verification key to check the validity of the signature.

5 [0024] Various aspects of what is described herein seek to provide a computer-implemented method using a chameleon hash function, wherein the chameleon hash function comprises one or more modular exponentiations and one or more modular multiplications. The method may include at a signature-verification phase: precomputing, at an offline phase of the signature-verification phase, each of the one or more modular
10 exponentiations of the chameleon hash function; and evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations.

[0025] In some aspects, the computer-implemented method may further include at a signature-generation phase: precomputing, at an offline phase of the signature-generation
15 phase, a standard signature on the chameleon hash function based on a random message and a randomness chosen randomly; signing, at an online phase of the signature-generation phase, for a received message using a secret key associated with the chameleon hash function to find a collision with the standard signature.

[0026] Both the signature-generation phase and the signature-verification phase may be
20 made faster by moving computation-intensive steps in the offline phase. The evaluation of the chameleon hash function at the signature-verification phase may require computing two (expensive) modular exponentiations modulo a large prime p - both the exponents are from Z_q for another large prime q whose size is comparable to that of p (to be precise, the bit-length of q is less than that of p by one). The cost of the evaluation of the chameleon hash

function in the signature-verification phase may predominate over the cost required for verifying the actual signature as both the modulus and the exponents may be very large.

[0027] A modular exponentiation may be typically performed using the square-and-multiply algorithm that checks each bit of the exponent sequentially and computes the successive squares (i.e., 2^i -th powers) of the base (and multiplies only when the corresponding bit is 1). Modular exponentiations may be done in a faster way with the help of precomputations for a fixed base. The chameleon hash function may require modular exponentiations with respect to two fixed bases g and h . Hence, when applied to this particular chameleon hash function, precomputations for each of these bases may be even more beneficial for faster evaluation of the hash function. As described herein, the verifier may precompute (and store) successive squares of the bases g and h in the offline phase of the signature-verification phase, such that evaluating the chameleon hash function in the online signature-verification phase may require only modular multiplications (no squaring) of some of these powers (depending on the bits set in the respective exponents) - which saves around two-third of the cost of the hash function evaluation (compared to that without precomputations) in the online phase.

[0028] The performance of the chameleon hash function evaluation with and without precomputations are compared. From the experimental results, it is observed that considerable gain is achieved in performance in case that the powers of the bases are precomputed and stored by the verifier. The precomputation at the offline signature-verification phase may be different from the precomputation at the offline signing phase, where the latter may be performed for each message (that is possibly obtained later in the online signing phase). On the other hand, the verifier may perform the offline signature-verification phase only one time and use the precomputed values to verify multiple (message, signature) pairs in the online signature-verification phase, unless the system

updates the chameleon hash function itself (to be specific, the base g or h present in the public key HK).

[0029] The techniques for speeding up modular exponentiations in the context of faster verification of the online/offline threshold signatures are provided herein. Specifically, an online/offline split in its signature-verification phase is provided. The verifier may precompute (and store) successive squares of the bases g and h in the offline phase of signature-verification phase, such that computing chameleon hash function evaluation in the online phase may require only modular multiplications of some of these powers (depending on the bits set in the exponents) - which may theoretically save around two-third of the cost of the hash function evaluation (compared to that without precomputations) in the online phase. From the experimental results, it is observed that there is a considerable gain in performance in case that the 2^i -th powers of the bases are precomputed and stored by the verifier. The offline phase of signature-verification may be performed only once (irrespective of the number of messages to sign) unless the system modifies the previously used chameleon hash function at some point of time.

[0030] There are other online/offline threshold signatures that also use discrete logarithm-based chameleon hash functions (e.g., use another variant of Pedersen's commitment which involves three modular exponentiations). The proposed techniques (i.e., precomputing the successive squares of the bases) for speeding up the online signature-verification phase can also apply for these schemes.

[0031] The following examples pertain to various aspects of the present disclosure.

[0032] Example 1 is a computer-implemented method using a chameleon hash function, wherein the chameleon hash function includes one or more modular exponentiations and one or more modular multiplications, the method including, at a signature-verification phase: precomputing, at an offline phase of the signature-verification phase, at least partially

each of the one or more modular exponentiations of the chameleon hash function; and evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations and one or more modular multiplications.

5 **[0033]** In Example 2, the subject matter of Example 1 may optionally include that precomputing, at the offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function comprising precomputing successive squares of a base of the respective modular exponentiation to obtain an array associated with the base of the respective modular exponentiation.

10 **[0034]** In Example 3, the subject matter of Example 2 may optionally include that an i -th element of the array associated with the base of the respective modular exponentiation is a 2^i -th powers of the base of the respective modular exponentiation.

[0035] In Example 4, the subject matter of Example 3 may optionally include that evaluating, at an online phase of the signature-verification phase, the chameleon hash
15 function comprises multiplying the i -th element of the array associated with the base of the respective modular exponentiation with a corresponding intermediate result iteratively obtained.

[0036] In Example 5, the subject matter of any one of Examples 1 to 4 may optionally include at a signature-generation phase: precomputing, at an offline phase of the signature-
20 generation phase, a standard signature on a value of the chameleon hash function when applied on a random message and a randomness chosen randomly; signing, at an online phase of the signature-generation phase, for a received message using a secret key associated with the chameleon hash function to find a collision with the value of the chameleon hash function precomputed at the offline phase of the signature-generation phase.

[0037] In Example 6, the subject matter of Example 5 may optionally include that the signature-generation phase comprises a signature-generation phase of a threshold signature scheme wherein up to a threshold number of secret keys in the threshold signature scheme are corrupted.

- 5 [0038] In Example 7, the subject matter of Example 2 may optionally include that the chameleon hash function is based on the discrete-logarithm problem, and is defined by a first modular exponentiation having a first base, a second modular exponentiation having a second base and a modular multiplication of the first and second modular exponentiations.

- [0039] In Example 8, the subject matter of Example 7 may optionally include that a
10 public key of the chameleon hash function is defined by the first base associated with the first modular exponentiation, the second base associated with the second modular exponentiation and a prime used for the first and second modular exponentiations and the modular multiplication.

- [0040] In Example 9, the subject matter of any one of Examples 1 to 8 may optionally
15 include that the precomputed one or more modular exponentiations is stored overhead at a verifier side.

[0041] In Example 10, the subject matter of any one of Examples 1 to 9 may optionally include that the chameleon hash function comprises discrete logarithm-based chameleon hash function comprising three modular exponentiations.

- 20 [0042] In Example 11, the subject matter of any one of Examples 1 to 10 may optionally include that precomputing, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function is performed one time by a single verifier.

- [0043] In Example 12, the subject matter of Example 11 may optionally include that
25 evaluating, at an online phase of the signature-verification phase, the chameleon hash

function based on the precomputed one or more modular exponentiations is performed multiple times by the single verifier.

[0044] In Example 13, the subject matter of Example 12 may optionally include: receiving a plurality of messages, evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations for each of the plurality of messages.

[0045] Example 14 is a computer program comprising instructions to execute the steps of the computer-implemented method of any one of Examples 1 to 13.

[0046] Example 15 is a non-transitory computer-readable medium having stored thereon the computer program of Example 14.

[0047] Example 16 is a system using a chameleon hash function, wherein the chameleon hash function comprises one or more modular exponentiations and one or more modular multiplications, the system including: a processor configured to, at a signature-verification phase: precompute, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function; and evaluate, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations.

[0048] In Example 17, the subject matter of Example 16 may optionally include at least one terminal device registered with the processor, the at least one terminal device configured to send the at least one message to an input module.

[0049] In Example 18, the subject matter of Example 17 may optionally include that the registration includes generating a unique identifier for each terminal device.

[0050] In Example 19, the subject matter of Example 17 may optionally include whitelisting specific IP addresses associated with the at least one terminal device.

[0051] In Example 20, the subject matter of Example 18 may optionally include that the at least one terminal device is a mobile computer device.

[0052] Unlike the key generation which is a one-time job and performed in the setup phase, signature-generation (Sign) or signature-verification (Verify) may be performed for
5 signing each message (or verifying each signature). Thus, efficiency of a signature scheme may depend on that of Sign and Verify. In the following, the proposed computer-implemented method at the signature-verification phase and the signature-generation phase will be described.

[0053] FIG. 1 is a flowchart showing an example computer-implemented method 100
10 using chameleon hash function, according to various embodiments of the present disclosure.

[0054] According to various non-limiting embodiments, the computer-implemented method 100 may use a chameleon hash function that includes one or more modular exponentiations and one or more modular multiplications. Chameleon hash functions, also called trapdoor-hash functions, are hash functions that feature a trapdoor that allows one to
15 find arbitrary collisions in the domain of the functions. However, chameleon-hash functions are collision resistant as long as the corresponding trapdoor (or secret key) is not known. Although chameleon hash function is used herein, the computer-implemented method 100 may use any function that may include one or more modular exponentiations and one or more modular multiplications and map data of arbitrary size to fixed-size values (e.g., hash
20 function).

[0055] According to various non-limiting embodiments, the computer-implemented method 100 may include, at a signature-verification phase: precomputing, at an offline phase of the signature-verification phase, each of the one or more modular exponentiations of the chameleon hash function. The one or more modular exponentiations of the chameleon hash
25 function may not depend on an input (e.g., a message) of the chameleon hash function.

[0056] In the context of various embodiments, an “offline phase” may refer to a phase or a time period or a time before a message is received or arrives. That may mean the traffic is low during the offline phase or the server performing the signature or verification is idle during the offline phase.

5 [0057] In the context of various embodiments, “precomputing” or “precomputation” may refer to the act of performing an initial computation before run time to generate a lookup table that can be used by an algorithm. The computer-implemented method 100 may include computing in advance each of the one or more modular exponentiations of the chameleon hash function and store the powers before a message arrives or is received.

10 [0058] According to various non-limiting embodiments, the computer-implemented method 100 may further include, at a signature-verification phase: evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations.

[0059] In the context of various embodiments, an “online phase” may refer to a phase
15 or a time period or a time when a message arrives or is received. That may mean the traffic is high during the online phase or the server performing the signature or verification is busy during the online phase.

[0060] In the context of various embodiments, the precomputed one or more modular exponentiations may include the lookup table that has been precomputed to avoid repeated
20 computation each time it is processed. The lookup table may include an array of powers that may be reused to speed up further processing. When a message arrives or is received, the computer-implemented method 100 may include evaluating, the chameleon hash function based on the array of powers that have been precomputed. The array of powers that have been precomputed may not rely on the message that arrives or is received. That is, for each

different message, the computer-implemented method 100 may include evaluating, the chameleon hash function based on the same array of powers that have been precomputed.

[0061] According to various non-limiting embodiments, the computer-implemented method 100 may include precomputing successive squares of a base of the respective modular exponentiation to obtain an array associated with the base of the respective modular exponentiation. Exponentiation may be defined by two numbers, the base b and the exponent or power n , and written as b^n . Accordingly, the successive squares of the base may compute b^n in a finite field GF (p). b^n may be represented as

$$\begin{aligned} b^n &= \prod_i \left(b^{\delta_i 2^i} \right) (\text{mod } p) \\ &= \prod_i \left(b^{\delta_i 2^i} (\text{mod } p) \right) (\text{mod } p), \end{aligned} \quad (1)$$

where $n = \delta_{w-1} \delta_{w-2} \dots \delta_1 \delta_0 = \delta_{w-1} 2^{w-1} + \delta_{w-2} 2^{w-2} + \dots + \delta_1 2^1 + \delta_0 2^0$ with $\delta_i \in \{0,1\}$ for each $i \in [0, w-1]$.

[0062] The term $b^{2^i} (\text{mod } p)$ corresponding to the item i in equation (1) may be computed by successive steps as:

$$\begin{aligned} &= b (\text{mod } p) = b_0 \\ &= b_0^2 (\text{mod } p) = b_1 \\ &= b_1^2 (\text{mod } p) = b_2 \\ &= b_2^2 (\text{mod } p) = b_4 \\ &= b_{i/2}^2 (\text{mod } p) = b_i \end{aligned} \quad (2)$$

[0063] Accordingly, a power n may be decomposed in successive powers of two as described above. An array associated with the base of the respective modular exponentiation may be obtained to store the values of the successive powers of the base b : $b_0, b_1, \dots, b_i, \dots, b_{w-1}$. An i -th element of the array associated with the base of the respective modular exponentiation may be a 2^i -th powers of the base of the respective modular exponentiation.

[0064] According to various non-limiting embodiments, the chameleon hash function, based on the discrete-logarithm problem, may be defined by a first modular exponentiation

having a first base, a second exponentiation having a second base and a modular multiplication of these two modular exponentiations. Given the message is m , the randomness is r , the first base is g , the second base is h and the multiplication is p , the chameleon hash function may be defined as $CH_{HK}(m, r) = g^r h^m \bmod p$. During the signature-
 5 verification phase, given the signature is σ (i.e. the tuple (σ, m, r)), the verifier may need to evaluate the chameleon hash function, i.e. whether $\text{Verify}_{pk}(CH_{HK}(m, r), \sigma) = \text{accept}$ or *reject*. pk is the public key for the underlying threshold signature scheme Σ_T . That is, the verifier may first compute the chameleon hash function $CH_{HK}(m, r) = g^r h^m \bmod p$ and then validate the signature σ on $CH_{HK}(m, r)$ using pk . In some embodiments, the chameleon hash
 10 function may include discrete logarithm-based chameleon hash function comprising three modular exponentiations.

[0065] According to various non-limiting embodiments, evaluating, at an online phase of the signature-verification phase, the chameleon hash function may include multiplying the i -th element of the array associated with the base of the respective modular
 15 exponentiation with a corresponding intermediate result iteratively obtained. More details will be described in the following with an exemplary algorithm.

[0066] According to various non-limiting embodiments, a public key may be defined by the first and second modular exponentiations and the modular multiplication. A verifier may evaluate the chameleon hash function using the public key, at the signature-verification
 20 phase. The precomputed one or more modular exponentiations may be stored overhead at a verifier side.

[0067] According to various non-limiting embodiments, precomputing, at an offline phase of the signature-verification phase, each of the one or more modular exponentiations of the chameleon hash function may be performed by a single verifier. Evaluating, at an
 25 online phase of the signature-verification phase, the chameleon hash function based on the

precomputed one or more modular exponentiations may be performed by multiple verifiers. That may mean that the one or more modular exponentiations of the chameleon hash function are the same and applicable to different verifiers. That is, precomputing the one or more modular exponentiations of the chameleon hash function at the offline phase of the signature-verification phase may be performed one time and the results (e.g., powers) may be used multiple times by different verifiers.

[0068] According to various non-limiting embodiments, the computer-implemented method 100 may further include: receiving a plurality of messages, evaluating, at the online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations for each of the plurality of messages. That may mean that the one or more modular exponentiations of the chameleon hash function are the same and applicable to different messages. That is, precomputing the one or more modular exponentiations of the chameleon hash function at the offline phase of the signature-verification phase may be performed one time and the results (e.g., powers) may be used multiple times for different messages.

[0069] According to various non-limiting embodiments, the precomputation (partial computation or complete computation) of one or more modular exponentiations required for chameleon hash function evaluation may be performed by a single (e.g., sole) verifier in an offline signature-verification phase at its end. The chameleon hash function evaluation, based on the precomputed (partially computed or completely computed) one or more modular exponentiations, may be performed by the single verifier in an online signature-verification phase at its end. The term of "single verifier" refers to such an individual verifier who individually performs both precomputation (offline) and chameleon hash evaluation (online) at its end. However, no "collision-finding" process for the chameleon hash function may be performed by that verifier in the online phase of the signature-verification phase at

its end. The verifier performs the offline phase of the signature-verification phase once and may perform the online phase of the signature-verification phase multiple times for multiple (message, signature) pairs. Note that there may be more than one such verifiers each of which may perform both the offline and online phases of the signature-verification phase multiple times for multiple (message, signature) pairs, at its end, without collaborating with others. On the contrary, both the online and offline phases of the signature-generation phase (as described herein) may need to involve multiple signers who work collaboratively to generate a signature.

[0070] According to various non-limiting embodiments, the computer-implemented method 100 may further include, at a signature-generation phase: precomputing, at an offline phase of the signature-generation phase, a standard signature on a hash value of the chameleon hash function when applied on a random message and a randomness chosen randomly; signing, at an online phase of the signature-generation phase, for a received message using a secret key associated with the chameleon hash function to find a collision with the hash value of chameleon hash function precomputed at the offline phase of the signature-generation phase. The computer-implemented method 100 may split the signature procedure (Sign) into two phases: the offline phase and the online phase. The offline phase of the signature-generation phase may not depend on the actual message to sign; thus, it may be performed offline (e.g., a signing server may perform this phase in its idle time when the traffic is low). On the other hand, the online phase may be performed once the signer gets the message and the corresponding signature is produced using the (intermediate) results obtained in the offline phase. Hash evaluation for the chameleon hash function may be collaboratively performed by multiple signers at the offline phase of the signature-generation phase; and collision-finding for the chameleon hash function may be collaboratively performed by multiple signers at the online phase of the signature-generation

phase. Splitting the signature-generation phase in two phases aims to speed up the online phase such that signatures may be generated very fast once messages are available.

- [0071] A "hash-sign-switch" paradigm in the context of online/offline signatures using a chameleon hash function may include a public key HK and a secret key (or trapdoor) TK .
- 5 The knowledge of the public key HK may allow anyone to evaluate the chameleon hash function $CH_{HK}(m, r)$ for a message $m \in M'$ and a randomness $r \in R$, where M' is the (input) message space and R is the randomness space for the hash function. On the other hand, the knowledge of the trapdoor TK may enable one to find collisions; that is, given the pair (HK, TK) , a message-randomness pair $(m_1, r_1) \in M' \times R$, and a message $m_2 \in M'$, another
- 10 randomness $r_2 \in R$ may be computed in polynomial time such that $CH_{HK}(m_1, r_1) = CH_{HK}(m_2, r_2)$. The standard signature scheme Σ may be split into an online/offline signature scheme as follows. In the offline phase, the signer may choose a message $a \in M'$ and a randomness $r \in R$ from the respective spaces uniformly at random and computes a standard signature σ (computed using Sign of Σ) on the hash value $CH_{HK}(a, r)$. Subsequently, in the online phase,
- 15 the signer may use the trapdoor TK to find a randomness value r' such that $CH_{HK}(m, r') = CH_{HK}(a, r)$, where m is the actual message to sign. Then, σ may serve as a signature on (m, r') (i.e., m may be thought of replacing the initial random value a). Accordingly, the cost of the online phase may be only that is required to find a collision. More efficient the collision finding is for a chameleon hash function, more is the speedup obtained in the online phase.
- 20 [0072] According to various non-limiting embodiments, the signature-generation phase may include a signature-generation phase of a threshold signature scheme wherein a threshold number of secret keys in the threshold signature scheme are corrupted. In a t -out-of- l threshold signature scheme, the signing key may be distributed among l signers, and up to t signers may be corrupted. In other words, any set of t (or less) signers may not produce

a valid signature on a message even if they collude with each other. Any threshold signature scheme Σ_T may be combined with a threshold version of a specific chameleon hash function (a variant of Pedersen commitment) based on the discrete-logarithm problem. The hash function may be best suited for using with the Lagrange interpolation required for reconstruction of a secret (or the evaluation of certain functions when applied on the secret) in a secret-sharing scheme.

[0073] According to various non-limiting embodiments, in the setup phase, a λ -bit safe prime $p = 2q+1$ is chosen such that q is also a prime. Let g' be a generator of the group $\mathbb{Z}_p^*$. Let $Q_p \subset \mathbb{Z}_p^*$ be the subgroup of quadratic residues which is generated by $g = (g')^2 \bmod p$ ($|Q_p| = (p-1)/2 = q$). A value $y \in \mathbb{Z}_q^*$ is chosen uniformly at random and set as the trapdoor TK . Given a message m and a randomness r , the chameleon hash function $CH_{HK}: \mathbb{Z}_q \times \mathbb{Z}_q \rightarrow Q_p$ is evaluated as $CH_{HK}(m, r) = g' h^m \bmod p$, where $HK = (p, g, h)$ is the public key and $TK = y$ is the trapdoor such that $h = g^y \bmod p$. Accordingly, both the message space M' and the randomness space R of the chameleon hash function CH_{HK} are $\mathbb{Z}_q$. On the other hand, the message space for the underlying threshold signature scheme Σ_T is $M = Q_p$, which is the output space of CH_{HK} .

[0074] According to various non-limiting embodiments, in the offline phase of the signature-generation phase, the signers may collaboratively compute $g^{r'} h^{m'} \bmod p$, where each signer may hold a share of m' and that of r' . The signers may then sign the value using the underlying threshold signature scheme (and shares of the signing key sk) and produce the signature σ . Subsequently in the online phase, given the actual message m to sign, any set of $(2t+1)$ signers finds an r collaboratively such that $CH_{HK}(m, r) = CH_{HK}(m', r')$. The distributed collision-finding process may be efficiently designed such that each of the signers has to perform only modular additions and modular multiplications (no expensive modular exponentiation may be required to be performed in the online phase).

[0075] An exemplary algorithm for the chameleon hash function evaluation will be now described by way of the following non-limiting examples. For simplicity, bit-lengths of both a randomness r and a message m may be assumed to be the same (for example, w). As $r, m \in \mathbb{Z}_q$, $1 \leq w \leq \lceil \log_2 q \rceil = \lambda-1$. Accordingly, $r = r_{w-1}r_{w-2} \dots r_1r_0$ and $m = m_{w-1}m_{w-2} \dots m_1m_0$,
 5 where $r_i, m_i \in \{0,1\}$ for each $i \in [0, w-1]$.

[0076] For comparison, the chameleon hash function evaluation is first done without precomputations using the pseudocode shown in Algorithm 1, where modular exponentiations are done using the standard square-and-multiply algorithm. In other words, Algorithm 1 shows a comparison example that at a signature-verification phase: when a
 10 message arrives or is received, evaluating the chameleon hash function that includes one or more modular exponentiations and one or more modular multiplications and no precomputation is performed. Specifically, the chameleon hash function in Algorithm 1 includes two modular exponentiations g^r and h^m modulo p , and one modular multiplication modulo p .

15 [0077] *Algorithm 1 Pseudocode for the evaluation of the chameleon hash function without precomputations*

Input: Public key $HK = (p, g, h)$, randomness r , message m

Output: $g^r h^m \bmod p$

$w \leftarrow \text{bit-length of } r \text{ and } m$

20 $b_1 \leftarrow g$

$b_2 \leftarrow h$

$a \leftarrow 1$

for i *from* 0 *to* $w-1$ *do*

if $r_i == 1$ *then*

25 $a = a * b_1 \bmod p$

```

    end if
     $b_1 = (b_1)^2 \bmod p$ 
    if  $m_i == 1$  then
     $a = a * b_2 \bmod p$ 
5  end if
     $b_2 = (b_2)^2 \bmod p$ 
  end for
  return  $a$ 

```

[0078] In Algorithm 1, the input includes a public key $HK = (p, g, h)$ for the chameleon hash function, a randomness r and a message m and the output includes a first modular exponentiations g^r , a second modular exponentiation h^m and a module multiplication p . Both r and m are assumed to be w -bit long. A first base b_1 is instantiated with g and a second base b_2 is instantiated with h . a is initialized to store intermediate results. When r_i bit is set, multiply the first base b_1 with the intermediate result a . The first base b_1 is updated by squaring its previous value. When m_i bit is set, multiply the second base b_2 with the intermediate result a . The second base b_2 is updated by squaring its previous value. Lastly, return the final result.

[0079] According to various non-limiting embodiments, the signature-verification may be split into online/offline phases as the successive (modular) squaring of the bases g and h (i.e., computing their 2^i -th powers) is done irrespective of the values of the bits of r and m . Accordingly, precomputing of the successive (modular) squaring of the bases g and h (i.e., computing their 2^i -th powers) may be performed even before getting the values r and m from the signers.

[0080] Consequently, a verifier may precompute and store the successive squares of the bases (i.e., their 2^i -th powers for $i \in [0, \lambda-2]$) modulo p in the offline phase (as shown in

Algorithm 2), and the verifier may only need to look up for the values of the corresponding powers while multiplying them with the intermediate result in the online phase (as shown in Algorithm 3). In addition, the verifier may have to verify the signature on the chameleon hash function $CH_{HK}(m, r)$.

5 **[0081]** *Algorithm 2 Pseudocode for precomputing 2^i -th powers of bases for $i \in [0, \lambda-2]$*

Input: Public key $HK = (p, g, h)$

Output: Arrays B_1 and B_2 storing 2^i -th powers of bases g and h (modulo p)

$b_1 \leftarrow g$

$b_2 \leftarrow h$

10 *for i from 0 to $\lambda-2$ do*

$B_1[i] = b_1$

$b_1 = (b_1)^2 \bmod p$

$B_2[i] = b_2$

$b_2 = (b_2)^2 \bmod p$

15 *end for*

return (B_1, B_2)

[0082] In Algorithm 2, the input includes a public key $HK = (p, g, h)$. A randomness r and a message m are not required at the offline phase. The output includes a first array B_1 and a second array B_2 storing 2^i -th powers of bases g and h (modulo p). A first base b_1 is
 20 instantiated with g and a second base b_2 is instantiated with h . Store 2^i -th power of g in the first array B_1 and store 2^i -th power of h in the second array B_2 . Base b_1 is updated by squaring its previous value and base b_2 is updated by squaring its previous value. Lastly, return the first and second arrays B_1, B_2 storing 2^i -th powers of g, h , respectively.

[0083] *Algorithm 3 Pseudocode for the evaluation of the chameleon hash function with*
 25 *precomputations*

Input: Public key $HK = (p, g, h)$, randomness r , message m , precomputed arrays B_1 and B_2

Output: $g^r h^m \bmod p$

$w \leftarrow \text{bit-length of } r \text{ and } m$

$a \leftarrow 1$

5 *for* i *from* 0 *to* $w-1$ *do*

if $r_i == 1$ *then*

$a = a * B_1[i] \bmod p$

end if

if $m_i == 1$ *then*

10 $a = a * B_2[i] \bmod p$

end if

end for

return a

[0084] In Algorithm 3, the input includes a public key $HK = (p, g, h)$, a randomness r , a
 15 message m and the precomputed first and second arrays B_1, B_2 storing 2^i -th powers of g, h .
 The output includes a first modular exponentiations g^r , a second modular exponentiation h^m
 and a module multiplication p . Both r and m are assumed to be w -bit long. a is initialized to
 store intermediate results. When r_i bit is set, multiply a precomputed array element $B_1[i]$
 with the intermediate result a . When m_i bit is set, multiply a precomputed array element
 20 $B_2[i]$ with the intermediate result a . Lastly, return the final result.

[0085] Under the assumption that half of the w bits in each of r and m are set, Algorithm
 1 may require a total of $3w$ modular multiplications (i.e., $1.5w$ for each modular
 exponentiation), $2w$ of them may be precomputed using Algorithm 2 at the offline phase of
 the signature-verification phase and, consequently, Algorithm 3 may require only w modular
 25 multiplications for the chameleon hash evaluation at the online phase of the signature-

verification phase. Accordingly, around 3-times speedup may be obtained for the chameleon hash evaluation.

[0086] The performance of the chameleon hash function evaluation with and without precomputations is measured. The codes are written in C and are compiled using GCC (version 7.5.0). To handle large integers and various operations on them, the GMP library (version 6.1.2) is used. The experiments are performed on a 3.60GHz Intel i7 processor with 12GB RAM.

[0087] In the experiments, two values of the security parameter λ is considered: 1024 and 2048. Depending on these values of λ , two primes p_1 and p_1' of 1024 bits and 2048 bits are set, respectively, where

$p_1 =$

174269982539036541301931268544932100716440383110960646231721351936086029
141078749569455742814085737299419818302338241343424835137289572566010975
414234807233508900596045765941761869098442362544148460362299965337814325
181983287322287646522763481667618747194380314651184549434306609618845767
472353211114250381787,

and

$p_1' =$

273980477976129782401740212941651348749406760543183866245457967652203857
547295172056299112948461704641169557563472708846462692227524381192833937
904417316339685902480379393189545277949061312021964149399710752310461250
842342316876209800969308621629515012588947795422123796307637282444793088
215690405975230506375382240047145421277385481635663238186532791315903054
906533694522969197907862510230196807504072034434487732480004449298457426
118068970093273090900533629679103162825760455814485064520918547708294268

567874914225795383653777656557404404701113352208597278572905297641336943
60974700428983551810139581506129814805099.

[0088] Algorithm 1 (without precomputations) and Algorithm 3 (with precomputations using Algorithm 2 and storing the powers) are implemented.

- 5 **[0089]** For $p=p_1$, the average time taken by the chameleon hash function evaluation (over ten random tuples of (g, y, r, m)) according to Algorithm 1 is 5.21 milliseconds, whereas that according to Algorithm 3 is 0.62 milliseconds over the same (g, y, r, m) tuples (one-time precomputation according to Algorithm 2 takes around 1.08 milliseconds over the same (g, y)).
- 10 **[0090]** For $p=p_1'$, the average time taken by the chameleon hash function evaluation (over ten random tuples of (g, y, r, m)) according to Algorithm 1 is 8.24 milliseconds, whereas that according to Algorithm 3 is 3.05 milliseconds over the same (g, y, r, m) tuples (one-time precomputation according to Algorithm 2 takes around 5.51 milliseconds over the same (g, y)).
- 15 **[0091]** From the above results, precomputations of the 2^i -th powers of the bases used in the chameleon hash function speed up its evaluation considerably. This is more significant with respect to efficiency, since these precomputations may need to be done only once in the lifetime of a chameleon hash function with a particular pair of bases (g, h) . Algorithm 1 as well as Algorithm 3 may be optimized in terms of codes.
- 20 **[0092]** The results using some other primes of lengths 1024 and 2048 bits are shown in Table 1.

Table 1: Time required (in milliseconds) for the chameleon hash evaluation without precomputations (using Algorithm 1) and with precomputations (using Algorithm 3)

Primes (bits)	Chameleon hash evaluation using Algorithm 1	Chameleon hash evaluation using Algorithm 3

p_2 (1024)	3.19	0.66
p_3 (1024)	4.56	0.77
p_4 (1024)	3.88	0.59
p_2' (2048)	8.68	3.04
p_3' (2048)	8.37	3.05
p_4' (2048)	8.40	3.05

[0093] The experiments may be run with multiple users in LAN and WAN settings in order to correctly evaluate the proposed algorithms. The proposed computer-implemented method may split the signature-verification phase into the offline phase of the signature-verification phase and the online phase of the signature-verification phase, wherein a single verifier precomputes the powers of the bases g and h in the offline phase and subsequently verifies the actual signature using these precomputations in the online phase. Thus, the gain in performance using precomputations (with respect to the same verifier verifying the same signature without any precomputations) is independent of the network settings (e.g., LAN or WAN settings). Moreover, as different signatures (possibly from different users) are generated by some subset of the same set of i signers, the public key HK remains the same, so do the bases g and h . Therefore, the performance gain in the chameleon hash evaluation (at the signature-verification phase) as well as the storage overhead due to precomputations of the powers of the bases may not depend on the number of signatures to be verified.

[0094] Unlike the online/offline signature-generation phase which requires communications among signing servers, the online/offline signature-verification phase, for a given signature, may be done by a single verifier without any communication needed.

[0095] For networks with low-latency communications (e.g., 5G-ultra-reliable low latency communications (URLLC) requires 1 to 5 milliseconds end-to-end latency guarantees), computation time may become the bottleneck. The online/offline signature-verification may be much helpful in this scenario. Moreover, precomputation done (one time) by the verifier may work for verifying multiple signatures.

[0096] Offline precomputations may speed up the chameleon hash function evaluation in the online signature-verification phase. However, it may come at the cost of storage overhead at the verifier side. For $p=p_1$, around 256 KB of storage may be required for storing successive powers of the bases g and h (128 KB per base). For $p=p_1'$, around 1 MB of storage
 5 may be required for storing successive powers of the bases g and h (512 KB per base). As the bases may be fixed at a time (in the lifetime of the chameleon hash function), so this may be the total storage overhead at the verifier for precomputations.

[0097] FIG. 2 is a flowchart showing an example offline phase of the computer-implemented method 200 using chameleon hash function, according to various
 10 embodiments of the present disclosure.

[0098] At step 201, given the public key $HK=(p, g, h)$ for the chameleon hash function with a large prime p and two bases g and h , the verifier V may compute 2^i -th powers of g and h at the offline phase of the signature-verification phase.

[0099] At step 202, the verifier V may populate two arrays B_1 and B_2 such that their i -th
 15 elements are the 2^i -th powers of g and h , respectively. The verifier V may store these arrays at its end.

[0100] At step 203, two-third of total modular multiplications required for the chameleon hash evaluation may be done in the offline phase of the signature-verification phase. Subsequently in the online signature-verification phase, the verifier V may retrieve
 20 the elements of the arrays B_1 and B_2 to evaluate the chameleon hash function.

FIG. 3 is a flowchart showing an example online phase of the computer-implemented method 300 using chameleon hash function, according to various embodiments of the present disclosure.

[0101] At step 301, given a message m , a randomness value r and the precomputed arrays B_1 and B_2 , the verifier V may evaluate the chameleon hash function and produce a hash value $h' = g^r h^m$ (modulo p) at the online phase of the signature-verification phase.

[0102] At step 302, given the hash value h' and the signature σ (signature on the pair
5 (m, r) using the signature-generation algorithm of the underlying threshold signature scheme Σ_T), the verifier V may verify the signature σ using the signature-verification algorithm of the threshold signature scheme Σ_T .

[0103] At step 303, only one-third of the total modular multiplications required for the chameleon hash evaluation may be done at the online phase of the signature-verification
10 phase (e.g., due to precomputing the arrays B_1 and B_2 in the offline phase and using them in the online phase).

[0104] Features (steps) that are described in the context of the computer-implemented method 200, 300 may correspondingly be applicable to the same or similar features (steps) in the computer-implemented method 100. Furthermore, additions and/or combinations
15 and/or alternatives as described for a feature (step) in the context of the computer-implemented method may correspondingly be applicable to the same or similar feature (step) in the computer-implemented method 100.

[0105] While the method described above is illustrated and described as a series of steps or events, it will be appreciated that any ordering of such steps or events are not to be
20 interpreted in a limiting sense. For example, some steps may occur in different orders and/or concurrently with other steps or events apart from those illustrated and/or described herein. In addition, not all illustrated steps may be required to implement one or more aspects or embodiments described herein. Also, one or more of the steps depicted herein may be carried out in one or more separate acts and/or phases.

[0106] A system using a chameleon hash function is also provided. The chameleon hash function may include one or more modular exponentiations and one or more modular multiplications. The system may include: a processor configured to, at a signature-verification phase: precompute, at an offline phase of the signature-verification phase, each
5 of the one or more modular exponentiations of the chameleon hash function; and evaluate, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations.

[0107] The system may further include at least one terminal device registered with the processor, the at least one terminal device configured to send the at least one message to the
10 input module. The registration may include generating a unique identifier for each terminal device. The system may further include whitelisting specific IP addresses associated with the at least one terminal device. The at least one terminal device may be a mobile computer device. The unique identifier may be associated with a device identity number of the mobile computer device.

15 [0108] A computer program comprising instructions to execute the steps of the computer-implemented method 100 is provided. The aforementioned instructions may include a computer program element or a computer program product. A non-transitory computer-readable medium having stored thereon the computer program is also provided.

[0109] FIG. 4 is a block diagram showing an example electronic device 400, according
20 to an embodiment of the present disclosure. The electronic device 400 may be a laptop computer, a desktop computer, a tablet computer, an automobile computer, a smart phone, a personal digital assistant, a server, or other electronic devices capable of running computer applications. In some embodiments, the electronic device 400 includes a processor 402, an input/output (I/O) module 404, memory 406, a power unit 408, and one or more network
25 interfaces 410. The electronic device 400 can include additional components. In some

embodiments, the processor 402, input/output (I/O) module 404, memory 406, power unit 408, and the network interface(s) 410 are housed together in a common housing or other assembly.

[0110] The processor 402 can execute instructions, for example, to generate output data based on data inputs. The instructions can include programs, codes, scripts, modules, or other types of data stored in memory (e.g., memory 406). Additionally or alternatively, the instructions can be encoded as pre-programmed or re-programmable logic circuits, logic gates, or other types of hardware or firmware components or modules. The processor 402 may be, or may include, a multicore processor having a plurality of cores, and each such core may have an independent power domain and can be configured to enter and exit different operating or performance states based on workload. Additionally or alternatively, the processor 402 may be, or may include, a general-purpose microprocessor, as a specialized co-processor or another type of data processing apparatus. In some cases, the processor 402 performs high-level operation of the electronic device 400. For example, the processor 402 may be configured to execute or interpret software, scripts, programs, functions, executables, or other instructions stored in the memory 406.

[0111] The example I/O module 404 may include a mouse, keypad, touch screen, scanner, optical reader, and/or stylus (or other input device(s)) through which a user of the electronic device 400 may provide input to the electronic device 400, and may also include one or more audio speakers for providing audio output and a video display device for providing textual, audiovisual, and/or graphical output.

[0112] The example memory 406 may include computer-readable storage media, for example, a volatile memory device, a non-volatile memory device, or both. The memory 406 may include one or more read-only memory devices, random-access memory devices, buffer memory devices, or a combination of these and other types of memory devices. In

some instances, one or more components of the memory can be integrated or otherwise associated with another component of the electronic device 400. The memory 406 may store instructions that are executable by the processor 402. In some examples, the memory 406 may store instructions for an operating system 412 and for application programs 414. The
5 memory 406 may also store a database 416.

[0113] The example power unit 408 provides power to the other components of the electronic device 400. For example, the other components may operate based on electrical power provided by the power unit 408 through a voltage bus or other connection. In some embodiments, the power unit 408 includes a battery or a battery system, for example, a
10 rechargeable battery. In some embodiments, the power unit 408 includes an adapter (e.g., an AC adapter) that receives an external power signal (from an external source) and converts the external power signal to an internal power signal conditioned for a component of the electronic device 400. The power unit 408 may include other components or operate in another manner.

[0114] The electronic device 400 may be configured to operate in a wireless, wired, or
15 cloud network environment (or a combination thereof). In some embodiments, the electronic device 400 can access the network using the network interface(s) 410. The network interface(s) 410 can include one or more adapters, modems, connectors, sockets, terminals, ports, slots, and the like. The wireless network that the electronic device 400 accesses may
20 operate, for example, according to a wireless network standard or another type of wireless communication protocol. For example, the wireless network may be configured to operate as a Wireless Local Area Network (WLAN), a Personal Area Network (PAN), a metropolitan area network (MAN), or another type of wireless network. Examples of
25 WLANs include networks configured to operate according to one or more of the 802.11 family of standards developed by IEEE (e.g., Wi-Fi networks), and others. Examples of

PANs include networks that operate according to short-range communication standards (e.g., BLUETOOTH®, Near Field Communication (NFC), ZigBee), millimeter wave communications, and others. The wired network that the electronic device 400 accesses may, for example, include Ethernet, SONET, circuit-switched networks (e.g., using
5 components such as SS7, cable, and the like), and others.

[0115] Some of the subject matter and operations described in this specification can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Some of the subject matter described in this
10 specification can be implemented as one or more computer programs, i.e., one or more modules of computer program instructions, encoded on a computer storage medium for execution by, or to control the operation of, data-processing apparatus. A computer storage medium can be, or can be included in, a computer-readable storage device, a computer-readable storage substrate, a random or serial access memory array or device, or a
15 combination of one or more of them. Moreover, while a computer storage medium is not a propagated signal, a computer storage medium can be a source or destination of computer program instructions encoded in an artificially generated propagated signal. The computer storage medium can also be, or be included in, one or more separate physical components or media (e.g., multiple CDs, disks, or other storage devices).

20 **[0116]** Some of the operations described in this specification can be implemented as operations performed by a data processing apparatus on data stored on one or more computer-readable storage devices or received from other sources.

[0117] The term “data processing apparatus” encompasses all kinds of apparatus, devices, and machines for processing data, including by way of example a programmable
25 processor, a computer, a system on a chip, or multiple ones, or combinations, of the

foregoing. The apparatus can include special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit). The apparatus can also include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol
5 stack, a database management system, an operating system, a cross-platform runtime environment, a virtual machine, or a combination of one or more of them.

[0118] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, declarative or procedural languages, and it can be deployed in any
10 form, including as a stand-alone program or as a module, component, subroutine, object, or other unit suitable for use in a computing environment. A computer program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program, or in multiple coordinated files (e.g.,
15 files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[0119] Some of the processes and logic flows described in this specification can be
20 performed by one or more programmable processors executing one or more computer programs to perform actions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[0120] While this specification contains many details, these should not be understood as limitations on the scope of what may be claimed, but rather as descriptions of features specific to particular examples. Certain features that are described in this specification or shown in the drawings in the context of separate embodiments can also be combined.

- 5 Conversely, various features that are described or shown in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable sub-combination.

- [0121] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular
10 order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can
15 generally be integrated together in a single product or packaged into multiple products.

[0122] A number of embodiments have been described. Nevertheless, it will be understood that various modifications can be made. Accordingly, other embodiments are within the scope of the following claims.

CLAIMS

What is claimed is:

1. A computer-implemented method using a chameleon hash function, wherein the chameleon hash function comprises one or more modular exponentiations and one or more modular multiplications, the method comprising, at a signature-verification phase:
precomputing, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function;
and
evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations and one or more modular multiplications.
2. The computer-implemented method according to claim 1, wherein precomputing, at the offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function comprising precomputing successive squares of a base of the respective modular exponentiation to obtain an array associated with the base of the respective modular exponentiation.
3. The computer-implemented method according to claim 2, wherein an i -th element of the array associated with the base of the respective modular exponentiation is a 2^i -th powers of the base of the respective modular exponentiation.
4. The computer-implemented method according to claim 3, wherein evaluating, at an online phase of the signature-verification phase, the chameleon hash function comprises

multiplying the i -th element of the array associated with the base of the respective modular exponentiation with a corresponding intermediate result iteratively obtained.

5. The computer-implemented method according to any one of claims 1 to 4, further comprising, at a signature-generation phase:

precomputing, at an offline phase of the signature-generation phase, a standard signature on a value of the chameleon hash function when applied on a random message and a randomness chosen randomly;

signing, at an online phase of the signature-generation phase, for a received message using a secret key associated with the chameleon hash function to find a collision with the value of the chameleon hash function precomputed at the offline phase of the signature-generation phase.

6. The computer-implemented method according to claim 5, wherein the signature-generation phase comprises a signature-generation phase of a threshold signature scheme wherein up to a threshold number of secret keys in the threshold signature scheme are corrupted.

7. The computer-implemented method according to claim 2, wherein the chameleon hash function is based on the discrete-logarithm problem, and is defined by a first modular exponentiation having a first base, a second modular exponentiation having a second base and a modular multiplication of the first and second modular exponentiations.

8. The computer-implemented method according to claim 7, wherein a public key of the chameleon hash function is defined by the first base associated with the first modular

exponentiation, the second base associated with the second modular exponentiation and a prime used for the first and second modular exponentiations and the modular multiplication.

9. The computer-implemented method according to any one of claims 1 to 8, wherein the precomputed one or more modular exponentiations is stored overhead at a verifier side.

10. The computer-implemented method according to any one of claims 1 to 9, wherein the chameleon hash function comprises discrete logarithm-based chameleon hash function comprising three modular exponentiations.

11. The computer-implemented method according to any one of claims 1 to 10, wherein precomputing, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function is performed one time by a single verifier.

12. The computer-implemented method according to claim 11, wherein evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations is performed multiple times by the single verifier.

13. The computer-implemented method according to claim 12, further comprising: receiving a plurality of messages,

evaluating, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations for each of the plurality of messages.

14. A computer program comprising instructions to execute the steps of the computer-implemented method of any one of claims 1 to 13.

15. A non-transitory computer-readable medium having stored thereon the computer program of claim 14.

16. A system using a chameleon hash function, wherein the chameleon hash function comprises one or more modular exponentiations and one or more modular multiplications, the system comprising:

a processor configured to, at a signature-verification phase:

precompute, at an offline phase of the signature-verification phase, at least partially each of the one or more modular exponentiations of the chameleon hash function; and

evaluate, at an online phase of the signature-verification phase, the chameleon hash function based on the precomputed one or more modular exponentiations.

17. The system according to claim 16, further comprises at least one terminal device registered with the processor, the at least one terminal device configured to send the at least one message to an input module.

18. The system according to claim 17, wherein the registration includes generating a unique identifier for each terminal device.

19. The system of claim 17, further comprises whitelisting specific IP addresses associated with the at least one terminal device.

20. The system of claim 18, wherein the at least one terminal device is a mobile computer device.