A Modular Framework for Multi-Factor Authentication and Key Exchange*

Nils Fleischhacker[1], Mark Manulis[2], and Amir Azodi[3]

[1] Saarland University, Germany
fleischhacker@cs.uni-saarland.de
[2] Surrey Centre for Cyber Security, University of Surrey, UK
mark@manulis.eu
[3] Hasso Plattner Institute, Germany
amir.azodi@hpi.de

Abstract. Multi-Factor Authentication (MFA), often coupled with Key Exchange (KE), offers very strong protection for secure communication and has been recommended by many major governmental and industrial bodies for use in highly sensitive applications. Over the past few years many companies started to offer various MFA services to their users and this trend is ongoing.

The MFAKE protocol framework presented in this paper offers *à la carte* design of multi-factor authentication and key exchange protocols by mixing multiple *types* and *quantities* of authentication factors in a secure way: MFAKE protocols designed using our framework can combine any subset of multiple low-entropy (one-time) passwords/PINs, high-entropy private/public keys, and biometric factors. This combination is obtained in a modular way from efficient single-factor password-based, public key-based, and biometric-based authentication-only protocols that can be executed in concurrent sessions and bound to a single session of an unauthenticated key exchange protocol to guarantee forward secrecy.

The modular approach used in the framework is particularly attractive for MFAKE solutions that require backward compatibility with existing single-factor authentication solutions or where new factors should be introduced gradually over some period of time. The framework is proven secure using the state-of-the art game-based security definitions where specifics of authentication factors such as dictionary attacks on passwords and imperfectness of the biometric matching processes are taken into account.

Keywords: two-factor, multi-factor authentication, tag-based authentication, key exchange, framework, modular design

1 Introduction

Authentication Factors An *authentication factor* is used to produce some evidence that an entity at the end of the communication channel is the one which it claims to be. Modern computer security knows different types of authentication factors, all of which are widely used in practice. Their standard classification considers three main groups (see e.g. [20]), characterized by the nature of provided evidence: knowledge, possession, and physical presence. The evidence of knowledge is typically offered by low-entropy *passwords*. These include memorizable (long-term) passwords or PINs, e.g. for login purposes and banking ATMs, and one-time passwords that are common to many online banking and e-commerce transactions. The evidence of possession corresponds to physical devices such as smart cards, tokens, or TPMs, equipped with long-term (high-entropy) *secret keys* and some cryptographic functionality. These devices have tamper-resistance to protect secret keys from exposure. The evidence of physical presence refers to unique *biometric identifiers* of human beings. These may include face profile, finger prints, iris images, or behavioral characteristics such as gait or voice, and are used in access control systems and many electronic passports.

* A shorter version of this paper appears in the proceedings of the 1st International Conference on Research in Security Standardisation (SSR 2014), December 16-17, 2014, Egham, UK. This is the full version.

A different approach might be needed for an attacker to compromise a particular factor, depending on its type and use. For instance, passwords are susceptible to social engineering (e.g. phishing) and dictionary attacks. Digital devices can be lost or stolen. Those offering tamper-resistance may nonetheless fall to reverse-engineering [21,22], side-channel attacks [19], and trojans (e.g. recent Sykipot Trojan attacks against smart cards). Biometric data can be obtained from a physical contact with the human or copied if available in a digital form. Since the number of personal biometrics that permit efficient use in security technologies is limited, their wide use across different application domains makes it even harder to keep those factors private.

Multi-Factor Authentication (with Key Exchange) The strength of Multi-Factor Authentication (MFA) is based on the assumption that if an entity has many authentication factors, regardless of their nature, then it is hard for the attacker to compromise them all. That is, by combining different factors within a single authentication process, MFA aims at higher assurance in comparison to single-factor schemes. MFA has found its way into practice[4], most notable are combinations of long-term passwords with secret keys, possibly stored in tokens (e.g. Two-Factor SSH with USB sticks) or any of these with one-time passwords (e.g. OATH HOTP/TOTP, RSA SecurID, Google Authenticator). Many companies, e.g. Google, Facebook, Yahoo are now offering their users optional two-factor authentication mechanisms based on one-time passwords. The increasing use of smart phones to access services and the recent progress by Apple and Samsung to equip smart phones with fingerprint readers is expected to further boost the practical deployment of the MFA technology. Since MFA is mostly used to authenticate a client/user to a remote server the authentication of the client becomes its main security goal. The server-side authentication in MFA protocols offers further protection and is typically performed without using multiple factors on the server side.

The concept of Multi-Factor Authenticated Key Exchange (MFAKE), formalized in [36], extends MFA with establishment of secure session keys. In addition to authentication goals it aims at key secrecy, usually modeled in terms of (Bellare-Rogaway style) AKE-security [8,10,16]. Earlier MFAKE protocols focused mostly on two factors and were often unsuccessful: for instance, password-token combination from [34] was broken in [40] which itself was broken in [31], the scheme from [37] was cryptanalyzed in [39], and a biometric-token combination from [32] has fallen in [33]. Partially, these attacks were due to the missing modeling and analysis in those works.

A formal approach to MFAKE introduced in [36] was the first to account simultaneously for all three types of authentication factors. Most notable is their modeling of biometric factors. Unlike some previous single-factor biometric schemes, e.g. [18,12], that regarded biometrics as low- or high-entropy secrets, [36] drops biometric secrecy in favor of the *liveness assumption* (see also [15,14]) aiming at physical presence of a user. The protocol from [36] has recently been cryptanalysed in [26], who showed how an adversary that steals user's password and impersonates the server can essentially compromise all other authentication factors of the client. The model in [36] didn't consider server authentication and the only way to prevent the above attack against the protocol is to require mandatory authentication on the server side. The protocol would remain insecure if server authentication is left optional (as intended by the model) due to the way in which client messages bind different authentication factors together, as also exploited in [26].

MFAKE protocols may differ not only in nature of factors but also in their quantity. To this end, [38] introduced Multi-Factor Password AKE (MFPAKE), extending the PAKE setting [7], where arbitrary many low-entropy passwords (long-term and one-time) can be combined to authenticate the client. Their protocol further offers public-key based server-side authentication and supports verifier-based PAKE setting from [23,24].

[4] MFA definitions and usage in practice are not consistent. For example, according to [1, Sec. 8.3], for two-factor authentication it suffices to deploy RADIUS authentication or use a single tamper-proof hardware token or a VPN access with individual certificate, whereas using two factors of the same type is not regarded as a two-factor solution. [2, Level 3] explicitly requires hardware tokens and some additional factor, e.g. password or biometric. This is in line with the perception of MFA where authentication with a certificate alone is considered single-factor [36] but deployment of two or more passwords multi-factor [38]. For the purpose of generality, we regard any approach with at least two factors irrespective of their type as MFA.

Generalized and Modular MFAKE Approach? Various problems in the design of secure MFAKE protocols, coupled with the fact that existing protocols differ in nature and quantity of deployed factors and that perception of MFA varies across products, standards, and research literature, motivates the need for a simpler and modular MFAKE approach.

Our goal is to build secure MFAKE protocols out of well-known and understood concepts behind existing single-factor solutions. We argue that in general this approach though not necessarily more efficient helps to avoid caveats, arising in the combination of factors and results in a cleaner, less error-prone protocol design. The generality of the approach can further be used to formally explain the relationships between MFAKE and single-factor authentication schemes, and its modularity is beneficial for an independent accommodation of other factors, e.g. for social authentication [13,17].

A general MFAKE protocol can be built from different types of single-factor AKE protocols that are then combined in a smart way into a secure MFAKE solution. The feasibility of this approach and its formal correctness is implied by our work. A direct combination of different black-box AKE schemes is sub-optimal since it would include some redundancy in the computations of forward-secure session keys. Therefore, our approach for a general MFAKE is to use single-factor *authentication-only* protocols (to avoid computation of multiple session keys) and derive one forward-secure session key at the end of the protocol.

1.1 Contributions and Organization

General MFAKE Model We introduce and model a general framework for (α, β, γ)-MFAKE, including its MFA-only version, building on the three-factor AKE model from [36]. In a standard client-server setting we admit arbitrary *quantities* and *combinations* of low-entropy passwords (long-term and one-time), high-entropy secret keys (possibly with corresponding public keys), and biometric factors (with explicit and implicit matching). We model dictionary attacks on passwords and also account for the imperfect matching process of biometric templates. When modeling biometrics we follow the *liveness assumption* of [36] and do not treat biometric distributions as secret. We discuss why this assumption is realistic from the practical point of view.

Remark 1. In Appendix A we further relate our (α, β, γ)-MFAKE framework to several existing authentication models and protocols. By varying the parameters α, β, and γ we can show that many current single-factor and multi-factor settings can be seen as special cases of our general framework: $(1, 0, 0)$-MFAKE implies PAKE models from [7,23], $(0, 1, 0)$-MFAKE implies two-party AKE models from [8,10], $(1, 1, 1)$-MFAKE subsumes the three-factor client authentication approach from [36], while $(\alpha, 0, 0)$-MFAKE is related to the MFPAKE protocol introduced in [38].

Modular (α, β, γ)-MFAKE Framework We give a simple generic (α, β, γ)-MFAKE protocol construction, based on sub-protocols that can be instantiated from a wide range of existing, well-understood and efficient authentication-only schemes. More precisely we consider arbitrary many independent runs of efficient authentication-only protocols that rely on passwords, secret keys, and biometrics and link them to a single independent session of an Unauthenticated Key Exchange (UKE) in a way that generically binds authentication and key establishment and results in an AKE-secure MFAKE protocol (with forward secrecy) that offers MFA for the client and strong (optional) authentication of the server.

To this end, we define a generalized notion of *tag-based* MFA, extending the preliminary concepts from [27] that considered the use of tags (auxiliary strings) in public key-based challenge-response scenarios. For all types of single-factor authentication-only protocols we demonstrate existence of efficient tag-based flavors and discuss their generic extensions with tags. We show how to use tags in an (α, β, γ)-MFAKE protocol to bind all independent (black-box) sub-protocols in a secure way. (In this way, for example, we avoid the type of problems identified in [26] for the protocol in [36].)

ORGANIZATION. Generalized (α, β, γ)-MFAKE, its MFA-only version, and security goals are modeled in Section 2. Our modular and generic construction of (α, β, γ)-MFAKE is specified and analyzed in Section 3, along with the underlying sub-protocols and their instantiations.

2 Generalized MFAKE: Definitions and Security

Our definitions of generalized MFAKE extend the model from [36], which in turn builds on the models from [8,7].

2.1 System Model and Correctness

Participants, Sessions, and Authentication Factors An MFAKE protocol is executed between two participants: a *client* C and a *server* S. Several instances of any participant can exist at a time. This models multiple concurrent protocol sessions. An instance of participant $\mathrm{U} \in \{C, S\}$ in session s is denoted as $[\mathrm{U}, s]$. The session id s is the transcript of all messages sent and received by the instance, except for the very last protocol message. At the end of the protocol each instance either accepts or rejects.

By $\mathsf{pid}([\mathrm{U}, s])$ we denote *partner identity* with which $[\mathrm{U}, s]$ is interacting in the protocol session. Two instances $[\mathrm{U}, s]$ and $[\mathrm{U}', s']$ are said to be *partnered* if and only if $\mathsf{pid}([\mathrm{U}, s]) = \mathrm{U}'$, $\mathsf{pid}([\mathrm{U}', s']) = \mathrm{U}$, and their session ids form *matching conversations* [8,10], denoted $s = s'$.

Each client C may have arbitrary types and quantities of authentication factors that it may use in multiple protocol sessions as detailed in the following.

PASSWORDS A client C may hold an array of α *passwords*, denoted $\boldsymbol{pwd}_C$. Each password $\boldsymbol{pwd}_C[i]$, $i = 1, \ldots, \alpha$ is assumed to have low entropy, chosen from a dictionary $\mathcal{D}_{\mathtt{pwd}}$. Passwords can be used across multiple sessions, in which case they are considered to be long-term. We also allow for one-time passwords [3,35] that have been previously registered with the server. Our setting can be extended to deal with verifier-based password authentication, e.g. [24,9,29], where the server stores some non-trivial function of $\boldsymbol{pwd}_C[i]$ for better protection against server compromise attacks.

CLIENT SECRET KEYS A client C may hold an array of β *secret keys*, denoted $\boldsymbol{sk}_C$. Each secret key $\boldsymbol{sk}_C[i] \in \mathtt{KeySp}$, $i = 1, \ldots, \beta$ is assumed to have high entropy. In case of public key-based client authentication there exists an array of corresponding public keys, denoted $\boldsymbol{pk}_C$, which is assumed to be known system-wide. Any $\boldsymbol{sk}_C[i]$ can be stored in a secure hardware token (e.g. in a smartcard or TPM), in which case its usage in the protocol assumes client's access to the corresponding device, i.e., our model doesn't distinguish between hardware tokens and private keys of the client.

BIOMETRICS For each client C there are γ public biometric distributions $\mathtt{Dist}_{C,i}$, $i = 1, \ldots, \gamma$. The process of measuring some biometric (being it face, any particular finger, or iris) is comprehended as drawing a *biometric template* $W_{C,i}$ according to $\mathtt{Dist}_{C,i}$. Upon the enrollment of the client an array $\boldsymbol{W}_C$ containing γ biometric templates $\boldsymbol{W}_C[i]$, $i = 1, \ldots, \gamma$ is created and will be used as a reference for the session-dependent matching process on the server's side. We do not need to require that $\boldsymbol{W}_C$ is stored in clear on the server's side. Our model admits the case, where the server stores some non-trivial transformation of $\boldsymbol{W}_C$, e.g. using secure sketches [18,12].

Functionality of biometric data matching is modeled through an algorithm $\mathsf{BioMatch}$, which takes as input a candidate template W^* and a reference template W, which may also be given *implicitly* in a transformed form, and outputs 1 indicating that W^* matches W and 0 otherwise. For example, $\mathsf{BioMatch}$ can require that the Hamming distance between W and W^* remains below some threshold, an approach used, e.g. in [12,36]. We also take into account that biometric measurements are *not* perfect:

- For any client C,

$$\Pr\left[\mathsf{BioMatch}(W^*_{C,i}, \boldsymbol{W}_C[i]) = 1 \mid W^*_{C,i} \leftarrow \mathtt{Dist}_{C,i}, i \in [1, \gamma]\right] \geq 1 - \mathsf{false}_i^{\mathrm{neg}},$$

  where $\mathsf{false}_i^{\mathrm{neg}}$ is the probability with which ith biometric of C is *falsely rejected.*
- For any two clients C', C with $C' \neq C$,

$$\Pr\left[\mathsf{BioMatch}(W^*_{C',i}, \boldsymbol{W}_C[i]) = 0 \mid W^*_{C',i} \leftarrow \mathtt{Dist}_{C',i}, i \in [1, \gamma]\right] \geq 1 - \mathsf{false}_i^{\mathrm{pos}},$$

  where $\mathsf{false}_i^{\mathrm{pos}}$ is the probability with which ith biometric of C' is *falsely accepted.*

While false rejection is important for MFA correctness, false acceptance impacts the lower bounds of the protocol's security.

Server Secret Key We assume that server S may have a high-entropy secret key sk_S with the corresponding system-wide known public key pk_S.

Generalized MFAKE We define generalized MFAKE and its correctness property.

Definition 1 ((α, β, γ)-MFAKE). *A multi-factor authenticated key exchange protocol* (α, β, γ)-$\mathsf{MFAKE}(C, S)$ *is a two-party protocol, executed between a client instance* $[C, s]$ *with* α *passwords,* β *secret keys, and* γ *biometric templates and a server instance* $[S, s']$ *such that at the end of their interaction each instance either accepts or rejects. The correctness property of the protocol requires that for all* $\kappa \in \mathbb{N}$*, if at the end of the protocol session* $[C, s]$ *accepts holding session key* k_C *and* $[S, s]$ *accepts holding session key* k_S*, and* $[C, s]$ *and* $[S, s]$ *are partnered, then* $\Pr[k_C = k_S] = 1$.

In *authentication-only* MFA protocols parties either reject or accept their communication partner *without* computing any session keys. The following definition of client's MFA towards the server accounts for imperfect biometric matching process, where servers may falsely reject clients.

Definition 2 ((α, β, γ)-MFA). *A multi-factor authentication-only protocol* (α, β, γ)-MFA *is a two-party protocol, executed between a client instance* $[C, s]$ *with* α *passwords,* β *secret keys, and* γ *biometric templates and a server instance* $[S, s']$ *such that at the end of their interaction the server instance either accepts* C *as its communication partner or rejects. Let '*$\mathtt{acc}\ C$*' denote the event that* $[S, s]$ *accepts the client. The correctness property of the* (α, β, γ)-MFA *protocol requires that* $\Pr[\mathtt{acc}\ C] \geq 1 - \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{neg}}$.

For *server-side authentication* the multi-factor aspect is typically irrelevant, i.e., the client decides whether to accept the server based on pk_S. The correctness property in this case is perfect.

2.2 Security Goals: AKE-Security and Mutual Authentication

MFAKE protocols must guarantee standard goals with respect to session key security and mutual authentication against any probabilistic polynomial-time adversary $\mathcal{A}$. Due to asymmetry with regard to the use of multiple factors on the client side and typically one factor (secret key) on the server's side, mutual authentication is dealt with separately for clients and servers.

Liveness Assumption for Biometrics We assume that biometric data is public and resort to liveness assumption [36] to ensure physical presence of a client. Liveness of a client C is modeled through a special *biometric computation oracle* $\mathsf{BioComp}([C, s], W_{\mathcal{A},i})$: depending on the state of $[C, s]$ this oracle uses client's secret keys $\boldsymbol{sk}_C$ and passwords $\boldsymbol{pwd}_C$ together with an input biometric template $W_{\mathcal{A},i}$ that must be chosen according to some *adversary-specified* distribution $\mathtt{Dist}_{\mathcal{A},i}$ to perform the required computation step that would otherwise be performed using a template $W^*_{C,i}$ chosen according to the distribution $\mathtt{Dist}_{C,i}$. The crucial condition here is that $\mathtt{Dist}_{\mathcal{A},i}$ must significantly differ from $\mathtt{Dist}_{C,i}$ such that $\Pr[\mathsf{BioMatch}(W_{\mathcal{A},i}, \boldsymbol{W}_C[i]) = 0] \geq 1 - \mathsf{false}_i^{\mathrm{pos}}$ for any $W_{\mathcal{A},i} \leftarrow_R \mathtt{Dist}_{\mathcal{A},i}$. For simplicity, we assume that $\mathcal{A}$ queries $\mathsf{BioComp}$ only with templates $W_{\mathcal{A},i}$ from the distributions $\mathtt{Dist}_{\mathcal{A},i}$, $1 \leq i \leq \gamma$ (alternative modeling of $\mathsf{BioComp}$ would require $\mathcal{A}$ to specify some template generation algorithm with a suitable distribution $\mathtt{Dist}_{\mathcal{A},i}$ which will be invoked within $\mathsf{BioComp}$ on each new query to pick $W_{\mathcal{A},i}$). Liveness assumption requires that any *new* message m, whose computation depends on the ith biometric template of C, must be previously generated by the $\mathsf{BioComp}$ oracle, before an active adversary can make use of it. Using $\mathsf{BioComp}$ oracle $\mathcal{A}$ can test own biometric templates in client's computations. Note that the liveness assumption allows for replay attacks on biometric-dependent messages, i.e. $\mathcal{A}$ can consult the $\mathsf{BioComp}$ oracle to obtain a new message in one session of C and then replay it in another.

Remark 2. Hao and Clarke [26] criticized [36] for the assumption that biometric data is public, arguing that templates that can be obtained by the adversary in practice are often of poor quality so that obtaining high-quality templates should be seen as a corruption of the client. This might be a valid argument in certain use cases, however, for the purpose of generality, it seems more appropriate to assume that biometric data is public and resort to the liveness assumption, when modeling security of biometric-based protocols. Since biometric data is used in many different domains (e.g. e-passports, personal computers, entry access systems, etc.) leakage of high-quality templates is not unlikely. In contrast to private keys, biometric characteristics are produced by nature and are bound to a specific person. From this perspective, their modeling via liveness assumption, aiming at user's *physical presence* seems to be more appropriate. Liveness assumption has also been in the focus of recent standardization initiatives, e.g. ISO/IEC WD 30107 Anti-Spoofing and Liveness Detection Techniques.

Client and Server Corruptions An active adversary $\mathcal{A}$ may corrupt authentication factors of a client C through its $\mathsf{CorruptClient}(C, \mathtt{type}, i)$ oracle by indicating the type of the corrupted factor and its position i in the array. Corrupted passwords and secret keys are revealed to $\mathcal{A}$, whereas corrupted biometric factors imply that $\mathcal{A}$ no longer needs to follow restrictions put forth by the liveness assumption on those factors. $\mathcal{A}$ can ask multiple $\mathsf{CorruptClient}$ queries for different factors of its choice. This models realistic scenarios, where different factors may require different attacks. Server corruptions are handled through the $\mathsf{CorruptServer}$ oracle, which responds with sk_S.

Adversarial Queries Our security definitions will be given in form of games with a PPT adversary $\mathcal{A}$ that interacts with the instances through a set of oracles, as specified in the following. We assume that $\mathrm{U}, \mathrm{U}' \in \{C, S\}$.

$\mathsf{Invoke}(\mathrm{U}, \mathrm{U}')$ allows $\mathcal{A}$ to invoke a session at party U with party U'. If U is a client then U' must be a server, and vice versa. In response, a new instance $[\mathrm{U}, s]$ with $\mathsf{pid}([\mathrm{U}, s]) = \mathrm{U}'$ is established. $[\mathrm{U}, s]$ takes as input the authentication factors of U. If $[\mathrm{U}, s]$ is supposed to send a message first then this message is generated and given to $\mathcal{A}$.

$\mathsf{Send}([\mathrm{U}, s], m)$ allows $\mathcal{A}$ to send messages to the protocol instances (e.g. by forwarding, modifying, or creating new messages). In general, the oracle processes m according to the state of $[\mathrm{U}, s]$ and eventually outputs the next message (if any) to $\mathcal{A}$. However, if $\mathrm{U} = S$ and m is such that it was not produced by an instance of $C = \mathsf{pid}([S, s])$ but its computation was expected to involve ith biometric of C, then m is processed only if it was output by $\mathsf{BioComp}([C, \cdot], W_{\mathcal{A},i})$ or if $\mathcal{A}$ previously queried $\mathsf{CorruptClient}(C, 3, i)$.

$\mathsf{BioComp}([C, s], W_{\mathcal{A},i})$ outputs message m (if any) computed based on the internal state of $[C, s]$ using $\boldsymbol{sk}_C$, $\boldsymbol{pwd}_C$, and $W_{\mathcal{A},i}$ (from $\mathtt{Dist}_{\mathcal{A},i}$ as explained above).

$\mathsf{RevealSK}([\mathrm{U}, s])$ gives $\mathcal{A}$ the session key computed by $[\mathrm{U}, s]$ (if such key exists).

$\mathsf{CorruptClient}(C, \mathtt{type}, i)$ allows $\mathcal{A}$ to corrupt authentication factors of C. If $\mathtt{type} = 1$ then $\mathcal{A}$ is given $\boldsymbol{pwd}_C[i]$; if $\mathtt{type} = 2$ then it receives $\boldsymbol{sk}_C[i]$; if $\mathtt{type} = 3$ then $\mathcal{A}$ receives nothing but the liveness assumption for the ith biometric of C is dropped.

$\mathsf{CorruptServer}(S)$ gives $\mathcal{A}$ server's S secret key sk_S.

Freshness The notion of freshness prevents $\mathcal{A}$ from using its oracles to attack the protocol in a trivial way. For instance, key secrecy and authentication goals will require that no protocol participant was fully corrupted during the protocol session: a *client C is fully corrupted* if and only if all existing authentication factors of C have been corrupted via corresponding $\mathsf{CorruptClient}(C, \cdot, \cdot)$ queries; a *server S is fully corrupted* if and only if a $\mathsf{CorruptServer}(S)$ query has been asked. Our definition of freshness aims at server instances since $\mathcal{A}$ will be required to break AKE-security for their session keys. This is not a limitation since protocol correctness guarantees that any accepted partnered client instance will compute the same key as the server instance. In protocols without server authentication $\mathcal{A}$ can impersonate the server and compute the same key as the client. An instance $[S, s]$ that has accepted is said to be *fresh* if all of the following holds:

- Upon acceptance of $[S, s]$ neither the server S nor the client $C = \mathsf{pid}([S, s])$ were fully corrupted.
- There has been no $\mathsf{RevealSK}$ query to $[S, s]$ or to its partnered client instance (if such instance exists).

The above conditions allow full corruption of parties after the session ends (upon acceptance) and thus capture the property of *forward secrecy* that is equally important for *all* types of authentication factors.

Remark 3. Freshness conditions can be made more complicated to incorporate specialized goals such as security against key compromise impersonation (KCI) and corruptions of ephemeral secrets (cf. [30] and its variants). These goals however are *factor-dependent*. For instance, $(\alpha, 0, 0)$-MFAKE protocols with *shared* passwords typically wouldn't offer KCI-security (which by definition makes sense only in the public key setting). It also seems unlikely that $(\alpha, 0, 0)$-MFAKE can tolerate leakage of ephemeral secrets (the only randomness used in the protocol) without enabling an offline dictionary attack. Our conditions thus offer a common security base for all (α, β, γ)-MFAKE flavors, without narrowing the possibility of extension towards more complex requirements.

Security of Session Keys Secrecy of session keys is modeled in terms of AKE-security in the Real-or-Random indistinguishability framework [4] where multiple Test queries that can be asked only to fresh instances $[S, s]$. Their answers depend on the value of bit b, which is fixed in the beginning of the game: if $b = 1$ then $\mathcal{A}$ receives the real session key held by $[S, s]$; if $b = 0$ then $\mathcal{A}$ is given a random key chosen uniformly from the set of all possible session keys. At the end of the game $\mathcal{A}$ outputs bit b' aiming to guess b. Let $\mathsf{Succ}_{\mathrm{AKE}}^{\mathcal{A},(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE}}(\kappa)$ denote the probability of the event $b' = b$ in a game played by $\mathcal{A}$ against the AKE-security of (α, β, γ)-MFAKE. Let q denote the total number of invoked sessions. (α, β, γ)-MFAKE is AKE-secure, if for all PPT adversaries $\mathcal{A}$ the following advantage is negligible in κ:

$$\mathsf{Adv}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) = \left|\mathsf{Succ}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q\Big(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{pos}}\Big) - \frac{1}{2}\right|.$$

AKE-security is relevant only for (α, β, γ)-MFAKE protocols from Definition 1. It doesn't apply to (α, β, γ)-MFA protocols from Definition 2 that do not support key establishment.

Authentication Requirements An (α, β, γ)-MFAKE protocol must further provide authentication, which we treat separately for clients and servers. A protocol which satisfies both offers *mutual authentication.*

Client Authentication. Let $\mathcal{A}$ be an adversary against client authentication of (α, β, γ)-MFAKE that interacts with client and server instances using the aforementioned queries (whereby Test queries are irrelevant). $\mathcal{A}$ breaks client authentication if there exists a server instance $[S, s]$ that has accepted a client $C = \mathsf{pid}([S, s])$, for which there exists no client instance that is partnered with $[S, s]$, and neither S nor C were fully corrupted upon the acceptance of $[S, s]$.

Let $\mathsf{Succ}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa)$ denote the success probability in breaking client authentication. The protocol is CAuth-secure, if for all PPT adversaries $\mathcal{A}$ the following advantage is negligible (in κ):

$$\mathsf{Adv}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) = \left|\mathsf{Succ}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q\Big(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{pos}}\Big)\right|.$$

This definition of CAuth-security is directly applicable to (α, β, γ)-MFA protocols from Definition 2. The advantage of $\mathcal{A}$ is denoted then $\mathsf{Adv}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFA},\mathcal{A}}(\kappa)$ and its success probability is subject to the same bounds as $\mathsf{Succ}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa)$. For (α, β, γ)-MFA protocols CAuth-security is the main property.

Remark 4. The low entropy of passwords and non-perfect biometric matching impose a lower bound $q(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{pos}})$ on the success probability of a CAuth-adversary. This bound is not imposed on the success probability with regard to server authentication as explained below.

Server Authentication. An adversary $\mathcal{A}$ against server authentication of (α, β, γ)-MFAKE interacts with client and server instances and breaks server authentication if there exists a client instance $[C, s]$ that has accepted a server $S = \mathsf{pid}([C, s])$, for which there exists no server instance that is partnered with $[C, s]$, and neither C nor S were fully corrupted upon the acceptance of $[C, s]$. (α, β, γ)-MFAKE is SAuth-secure, if for all PPT adversaries $\mathcal{A}$ the probability of breaking server authentication, denoted $\mathsf{Succ}_{\mathrm{SAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa)$ is negligible in the security parameter κ.

3 Modular Design of MFAKE Protocols

Our general (α, β, γ)-MFAKE protocol is built in a modular way from sub-protocols for different authentication factors, yet with some extensions and optimizations. We start with the main building blocks.

3.1 Tag-based Authentication

Tag-based Authentication (TbA) [27] accounts for the use of auxiliary, possibly public, strings (tags) in authentication protocols. In TbA each party uses a tag, in addition to the authentication factor, and the protocol guarantees that if parties accept then their tags match. For instance, the server accepts some client in a session if and only if that client was alive during that session and used as input the same tag as the server. For public key-based challenge-response protocols, [27] gave a signature-based compiler with the TbA property. In our work we require a more general TbA notion that in addition to public keys encompasses passwords and biometrics as defined in the following.

Definition 3 (Tag-based MFA). *A tag-based MFA protocol (α, β, γ)-tMFA is an (α, β, γ)-MFA protocol from Definition 2, where in addition the client instance $[C, s]$ takes as input tag t_C, the server instance $[S, s]$ takes as input tag t_S, and if $t_C \neq t_S$ then both parties reject; otherwise, they accept as in the (α, β, γ)-MFA protocol.*

Tag-based CAuth-security: *Let $\mathcal{A}$ be a PPT adversary against client authentication of (α, β, γ)-tMFA that interacts with the instances of C and S using the same oracles as for (α, β, γ)-MFA, except that the* Invoke *oracle is modified such that it receives tag t as an additional input from $\mathcal{A}$. $\mathcal{A}$ is said to break* CAuth-security *of (α, β, γ)-tMFA if at the end of its interaction there exists a server instance $[S, s]$ that was invoked with tag t_S and has accepted a client $C = \mathsf{pid}([S, s])$, for which there exists no client instance that was invoked with tag $t_C = t_S$ and is partnered with $[S, s]$, and neither S nor C were fully corrupted upon the acceptance of $[S, s]$. The corresponding advantage of $\mathcal{A}$, denoted $\mathsf{Adv}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{tMFA},\mathcal{A}}(\kappa)$, is then defined analog to the advantage in (α, β, γ)-MFA.*

$\mathcal{A}$ is allowed to test tags of its own choice, i.e. existence of a partnered client instance that was invoked with a tag $t_C \neq t_S$ leads to a successful attack. Definitions of *tag-based server authentication* in (α, β, γ)-tMFA and success probability $\mathsf{Succ}_{\mathrm{SAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{tMFA},\mathcal{A}}(\kappa)$ are obtained by reversing the roles of C and S, as for (α, β, γ)-MFA in Section 2.2.

3.2 Utilized Sub-Protocols and Their Examples

Our framework constructs (α, β, γ)-MFAKE in a modular way from simpler protocols that represent special cases of tag-based MFA. We first describe corresponding (non tag-based) protocols for authentication and provide some examples, including the discussion on how to extend those protocols with tags.

PwA **: (Tag-based) Password-based Authentication Protocol** The first sub-protocol is for password-based authentication, denoted PwA, in which only one party (in our case the client) authenticates itself to the other party (server). In our generalized MFA model the adversarial advantage against client authentication of PwA becomes $\mathsf{Adv}_{\mathrm{CAuth}}^{\mathcal{A},\mathsf{PwA}}(\kappa) = \mathsf{Adv}_{\mathrm{CAuth}}^{\mathcal{A},(1,0,0)\text{-}\mathsf{MFA}}(\kappa)$.

For instance, an AKE-secure PAKE protocol with key confirmation from client to server, which is proven secure in the model from [7] can be used as PwA. On the other hand, those PAKE protocols can be somewhat simplified since we do not require PwA to provide session keys. The following is an example for the PAKE protocol from [6] when only client-side authentication with key confirmation is applied.

PwA EXAMPLE. Let $(\mathbb{G}, g, q)$ be a description of the cyclic group of prime order q with generator g that together with two elements $V, W \in \mathbb{G}$ and a hash function $\mathcal{H} : \{0,1\}^* \mapsto \{0,1\}^\kappa$ build public parameters. Assume that $pwd \in \mathbb{Z}_q$ is shared between C and S. In a PwA session, derived from [6], S sends $Y^* = g^y W^{pwd}$ for some $y \leftarrow_R \mathbb{Z}_q$ to C. C picks $x \leftarrow_R \mathbb{Z}_q$ and responds with $(X^*, h) = (g^x V^{pwd}, \mathcal{H}(C, S, Y^*, X^*, (Y^*/W^{pwd})^x,$

$pwd)$). S checks whether $h = \mathcal{H}(C, S, Y^*, X^*, (X^*/V^{pwd})^y, pwd)$ and accepts the client in this case. It is easy to see that client authentication of this PwA follows from the security of PAKE in [6].

Verifier-based PwA. The Ω-method introduced in [24,23] transforms any PAKE into a verifier-based (aka. asymmetric or augmented) PAKE where passwords are stored on the server side in a blinded way using a random oracle $\mathcal{H}'$, a symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and an additional pair of signing keys (sk, pk), which are not treated as an authentication factor. For a given password pwd the server stores $(\mathcal{H}'(pwd), \mathsf{Enc}_{pwd}(sk))$. The Ω-method proceeds as follows. First a (symmetric) PAKE session is executed using $\mathcal{H}'(pwd)$ as a password on both sides, resulting in an intermediate PAKE key k. This key is used to derive two independent keys k' and k'' and the client is given $\mathsf{Enc}_{k'}(\mathsf{Enc}_{pwd}(sk))$. C decrypts sk and sends a signature on the entire protocol transcript. If this signature verifies using pk the server accepts the client. The session key of verifier-based PAKE becomes k''. The Ω-method can be applied to obtain verifier-based PwA from plain PAKE protocols, in which case k'' can be omitted.

Tag-based (verifier-based) PwA. In the symmetric case any PwA protocol can be transformed into a tag-based tPwA as follows. Parties on input their tags t first compute $\mathcal{H}_T(pwd, t)$ using a cryptographic hash function $\mathcal{H}_T$, which then serves as a password for the original PwA. Since in Definition 3 the adversary specifies tags upon invocation of an instance any successful CAuth-adversary against tPwA can either be used to break CAuth-security of PwA or to find a collision for $\mathcal{H}$, i.e. $\mathsf{Adv}^{\mathsf{tPwA},\mathcal{A}}_{\mathrm{CAuth}}(\kappa) \leq \mathsf{Adv}^{\mathsf{PwA},\mathcal{A}}_{\mathrm{CAuth}}(\kappa) + q\epsilon_{\mathcal{H}_T}(\kappa)$ in q protocol sessions. A similar trick can be applied to verified-based PwA constructed using the aforementioned Ω-method — instead of $\mathcal{H}'(pwd)$ in the initial (symmetric) PwA session parties would use $\mathcal{H}_T(\mathcal{H}'(pwd), t)$. Security of such verifier-based tPwA follows from the security of the underlying PwA, the Ω-method, and the collision-resistance of $\mathcal{H}_T$.

PkA: (Tag-based) Public Key Authentication Protocol The second sub-protocol is a single-side authentication protocol in the public key setting, denoted PkA, with adversarial advantage against its client authentication defined as $\mathsf{Adv}^{\mathcal{A},\mathsf{PkA}}_{\mathrm{CAuth}}(\kappa) = \mathsf{Adv}^{\mathcal{A},(0,1,0)\text{-}\mathsf{MFA}}_{\mathrm{CAuth}}(\kappa)$.

Tag-based PkA. Examples of PkA include challenge-response protocols, where S sends a (high-entropy) challenge r to C, and C replies with a function of its secret key, e.g. a signature. A generic extension of such PkA protocols with tags, denoted tPkA, uses a cryptographic hash function $\mathcal{H}_T$ and follows immediately from [27] — the challenge r received by C with tag t_C is transformed into $r'_C = \mathcal{H}_T(r, t_C)$, which is then used to generated response to S where it is verified using $r_S = \mathcal{H}_T(r, t_S)$. As shown in [27] this conversion is applicable to various classes of PkA protocols.

BiA: (Tag-based) Biometric-based Authentication Protocol The third sub-protocol is a biometric-based authentication protocol, denoted BiA, in which C authenticates towards S that holds some (possibly blinded) reference template of C. In line with our model (and [36]) we work with public biometric factors and denote the adversarial advantage against client authentication of BiA as $\mathsf{Adv}^{\mathcal{A},\mathsf{BiA}}_{\mathrm{CAuth}}(\kappa) = \mathsf{Adv}^{\mathcal{A},(0,0,1)\text{-}\mathsf{MFA}}_{\mathrm{CAuth}}(\kappa)$.

Tag-based BiA example. Let $(\mathbb{G}, g, q)$ be a cyclic group of sufficiently large prime order q. C and S first execute an unauthenticated Diffie-Hellman key exchange in $\mathbb{G}$ by exchanging g^x and g^y. Consider two hash functions $\mathcal{H}_1, \mathcal{H}_2 : \mathbb{G} \mapsto \{0,1\}^\kappa$. Let $W'_{C,i}$ resp. $W_{C,i}$ denote the ith bit of the corresponding template. For each bit i the client computes $h_i = \mathcal{H}_1(g^x, g^y, g^{xy}, W'_{C,i}, i)$ using its version of g^{xy} and sends the resulting set $\{h_i\}_i$ to S. S re-computes corresponding values using its version of g^{xy} and the reference template W_C, and accepts the client if τ or more hash values from $\{h_i\}_i$ match. Note that if liveness assumption is in place then the adversary is prevented from sending any h_i that was not computed beforehand through the $\mathsf{BioComp}$ oracle. The tag-based CAuth-security of the protocol follows then directly from the classical CDH assumption in the random oracle model.

UKE: Unauthenticated Key Exchange Observe that tag-based authentication protocols do not offer computation of session keys. In our modular (α, β, γ)-MFAKE protocol we will use an *unauthenticated* key exchange, denoted UKE, as another sub-protocol. We assume that UKE satisfies the following standard definition (see e.g. [27]) tailored to the client-server scenario.

Definition 4 (Unauthenticated KE). *An unauthenticated key exchange protocol, denoted* UKE*, is a two-party protocol executed between a client instance* $[C, s]$ *and a server instance* $[S, s']$ *such that at the end both instances accept holding respective session keys* k_C *and* k_S *or reject. Let* $s = \mathsf{tr}_C$ *and* $s' = \mathsf{tr}_S$ *be respective communication transcripts of the two instances. An* UKE *protocol is correct if their partnering, i.e.* $s = s'$*, implies equality of their session keys, i.e.,* $k_C = k_S$*.*

KE-SECURITY. *Consider the following attack game against some correct* UKE *protocol: A PPT adversary* $\mathcal{A}$ *receives as input the security parameter* κ *and can query the* $\mathsf{Transcript}$ *oracle which is parameterized with a random bit* b *fixed in the beginning of the game. On an ith query the* $\mathsf{Transcript}$ *oracle executes a protocol session between two new instances of* C *and* S*, and hands its communication transcript* tr_i *and a key* k_i *to* $\mathcal{A}$*, where* k_i *is real if* $b = 0$ *or randomly chosen (for each new* $\mathsf{Transcript}$ *query) if* $b = 1$*. At some point* $\mathcal{A}$ *outputs bit* b'*. An* UKE *protocol is KE-secure if the following advantage is negligible in* κ *for all* $\mathcal{A}$*:*

$$\mathsf{Adv}_{\mathrm{KE}}^{\mathsf{UKE},\mathcal{A}} = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

UKE EXAMPLE. The unauthenticated Diffie-Hellman key exchange protocol in a cyclic group $(\mathbb{G}, g, q)$, where C and S exchange g^x and g^y, respectively, and derive their session keys via $\mathcal{H}(g^x, g^y, g^{xy})$ offers a straight-forward KE-secure UKE scheme in the random oracle model under the CDH assumption.

3.3 Modular (α, β, γ)-MFAKE Protocol Framework

We now detail the modular design of a generalized (α, β, γ)-MFAKE protocol, which supports arbitrary combinations of authentication factors, both in type and quantity. In addition to the sub-protocols from the previous section, its construction utilizes four hash functions $\mathcal{H}_T, \mathcal{H}_C, \mathcal{H}_S, \mathcal{H}_k : \{0,1\}^* \mapsto \{0,1\}^\kappa$, modeled as random oracles that are used for the purpose of tag derivation, key confirmation, and key derivation.

PROTOCOL DESCRIPTION. (α, β, γ)-MFAKE is built from *four* sub-protocols: UKE, tPwA, tPkA, and tBiA. The design is based on the following idea (see also Figure 1): first, C and S run one UKE session resulting in unauthenticated session keys k_0 for the client and k'_0 for the server, that are then used by both parties to derive tags (t_C and t_S) through $\mathcal{H}_T$. Then, an appropriate tag-based sub-protocol is executed independently for each authentication factor of the client. C and S thus execute α sessions of tPwA, β sessions of tPkA (with client-side authentication), and γ sessions of tBiA, possibly in parallel. S aborts the protocol and rejects C if any of those sessions results in the rejection of the client. The server authentication is optional and is executed through a session of tPkA (with server-side authentication). After finishing all sub-protocols C and S hold their so-far transcripts $\{\mathsf{tr}_i\}_{i=0,\ldots,\alpha+\beta+\gamma+1}$ and $\{\mathsf{tr}'_i\}_{i=0,\ldots,\alpha+\beta+\gamma+1}$, respectively, and proceed with the confirmation: C sends a hash value, computed with $\mathcal{H}_C$, on input its unauthenticated key material from the UKE session and session identifier s, which comprises its so-far transcripts and the identities of both parties. S verifies that this hash value is as expected. For the optional server authentication, S responds with its own hash value, computed using $\mathcal{H}_S$ on similar inputs as in the client's case. Upon successful confirmation parties accept with session keys k_C resp. k_S, derived using $\mathcal{H}_k$.

INSTANTIATIONS. Our general (α, β, γ)-MFAKE protocol can be instantiated using concrete sub-protocols from Section 3.2. That is, working in prime-order cyclic groups $(\mathbb{G}, g, q)$, we can use unauthenticated Diffie-Hellman key exchange for UKE, a tag-based password-based authentication protocol PwA obtained from the PAKE protocol in [6] (as detailed in Section 3.2), a suitable tag-based challenge-response protocol for tPkA, e.g. using DSS or Schnorr signatures, and our simple tBiA protocol with explicit matching based on the Hamming distance mentioned in Section 3.2. By using the Ω-method from [24] (as also discussed in Section 3.2) we can obtain a verifier-based version of tPwA and use it in our construction. Finally, as evident from the security analysis in Section 3.4, (α, β, γ)-MFAKE can be instantiated from arbitrary sub-protocols as long as those satisfy the required authentication goals. Moreover, as discussed in Section 3.2, tPwA can be obtained generically from PwA, and for a large class of PkA there exists a generic conversion to tPkA. Hence,

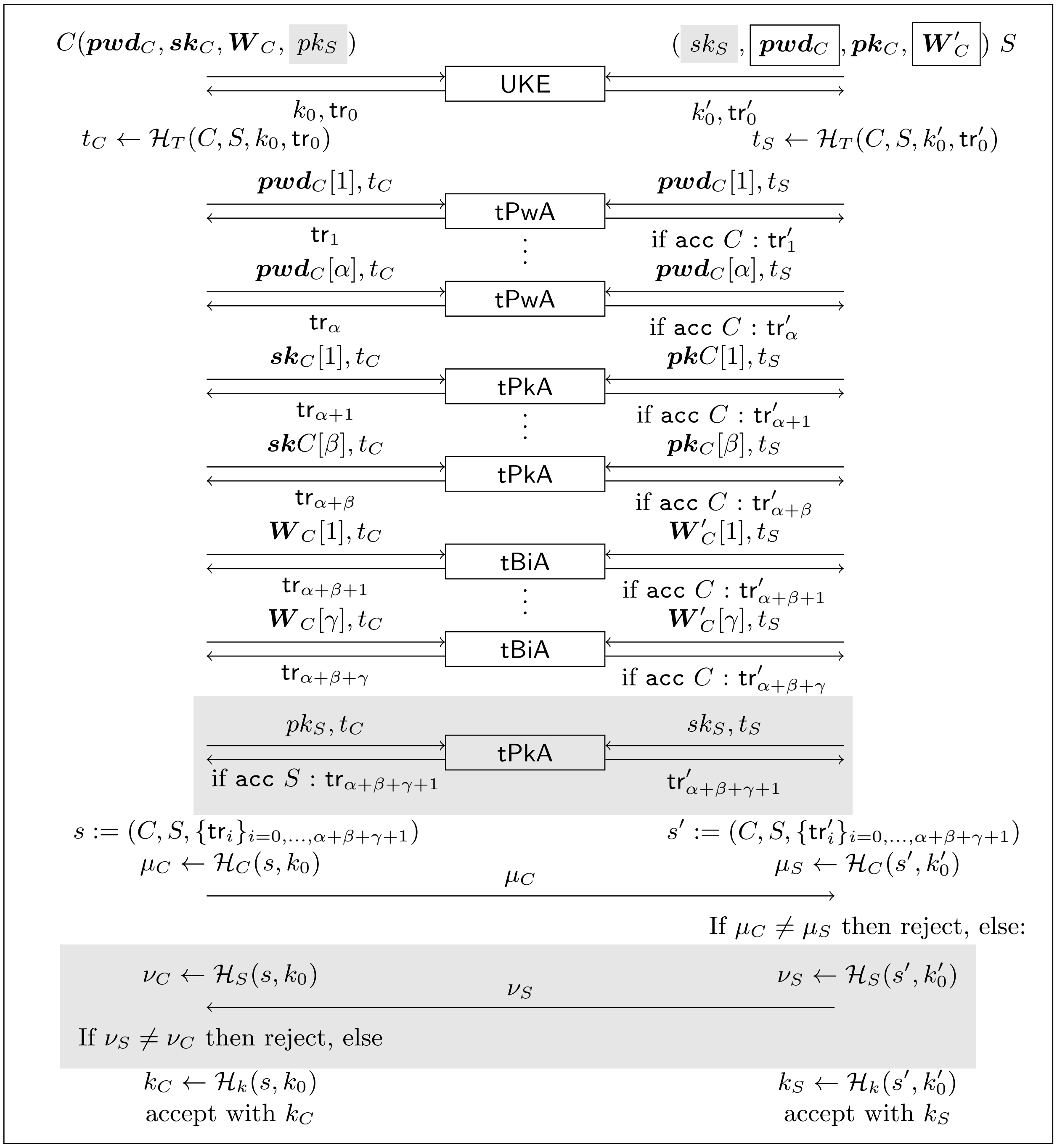
$C(\boldsymbol{pwd}_C, \boldsymbol{sk}_C, \boldsymbol{W}_C, pk_S)$
$(sk_S, \boxed{\boldsymbol{pwd}_C}, \boldsymbol{pk}_C, \boxed{\boldsymbol{W}'_C})\ S$
UKE
k_0, tr_0
k'_0, tr'_0
$t_C \leftarrow \mathcal{H}_T(C, S, k_0, \mathsf{tr}_0)$
$t_S \leftarrow \mathcal{H}_T(C, S, k'_0, \mathsf{tr}'_0)$
$\boldsymbol{pwd}_C[1], t_C$
$\boldsymbol{pwd}_C[1], t_S$
tPwA
tr_1
if acc C : tr'_1
$\boldsymbol{pwd}_C[\alpha], t_C$
$\boldsymbol{pwd}_C[\alpha], t_S$
tPwA
tr_α
if acc C : tr'_α
$\boldsymbol{sk}_C[1], t_C$
$\boldsymbol{pk}C[1], t_S$
tPkA
$\mathsf{tr}_{\alpha+1}$
if acc C : $\mathsf{tr}'_{\alpha+1}$
$\boldsymbol{sk}C[\beta], t_C$
$\boldsymbol{pk}_C[\beta], t_S$
tPkA
$\mathsf{tr}_{\alpha+\beta}$
if acc C : $\mathsf{tr}'_{\alpha+\beta}$
$\boldsymbol{W}_C[1], t_C$
$\boldsymbol{W}'_C[1], t_S$
tBiA
$\mathsf{tr}_{\alpha+\beta+1}$
if acc C : $\mathsf{tr}'_{\alpha+\beta+1}$
$\boldsymbol{W}_C[\gamma], t_C$
$\boldsymbol{W}'_C[\gamma], t_S$
tBiA
$\mathsf{tr}_{\alpha+\beta+\gamma}$
if acc C : $\mathsf{tr}'_{\alpha+\beta+\gamma}$
pk_S, t_C
sk_S, t_S
tPkA
if acc S : $\mathsf{tr}_{\alpha+\beta+\gamma+1}$
$\mathsf{tr}'_{\alpha+\beta+\gamma+1}$
$s := (C, S, \{\mathsf{tr}_i\}_{i=0,\dots,\alpha+\beta+\gamma+1})$
$s' := (C, S, \{\mathsf{tr}'_i\}_{i=0,\dots,\alpha+\beta+\gamma+1})$
$\mu_C \leftarrow \mathcal{H}_C(s, k_0)$
$\mu_S \leftarrow \mathcal{H}_C(s', k'_0)$
μ_C
If $\mu_C \neq \mu_S$ then reject, else:
$\nu_C \leftarrow \mathcal{H}_S(s, k_0)$
$\nu_S \leftarrow \mathcal{H}_S(s', k'_0)$
ν_S
If $\nu_S \neq \nu_C$ then reject, else
$k_C \leftarrow \mathcal{H}_k(s, k_0)$
accept with k_C
$k_S \leftarrow \mathcal{H}_k(s', k'_0)$
accept with k_S

Fig. 1. (α, β, γ)-MFAKE Protocol. The inputs sk_S and pk_S are optional for the case of server authentication and so is the server-authenticated execution of tPkA and the confirmation message ν_S. These optional parts are shown with a light gray background. Boxed input $\boldsymbol{pwd}_C$ on the server side reflects that client's passwords could be stored in some blinded way, in which case tPwA is assumed to follow the steps from [24,9,29]. Boxed input $\boldsymbol{W}'_C$ on the server side means that client's reference templates are not necessarily stored in clear, in which case tBiA must provide implicit matching functionality.

all building blocks of (α,β,γ)-MFAKE can be realized using existing efficient (single-factor) authentication solutions.

Performance optimizations. The only dependency amongst the different black-box runs of tag-based authentication sub-protocols is the input tag obtained after the UKE session. Therefore, all subsequent sub-protocol runs can be parallelized, resulting in three generic rounds (UKE, tag-based sub-protocols, and confirmation round). Of course, care should be taken to match client and server messages within each round, in order to account for the potential mismatch in the sending and delivery order of messages in parallel sub-protocol sessions. This can be done by pre-pending labels indicating that a message belongs to the ith session and using these labels to construct matching transcripts on both sides. Further optimizations may include interleaving of messages and using one random challenge of S for all β sessions of tPkA and another one for all γ sessions of tBiA, resulting in a three-pass protocol for MFA-based client authentication and five-pass protocol with further authentication of the server.

3.4 Security Analysis

The initial UKE execution contributes to the forward secrecy of the session keys. In particular, successful key confirmation guarantees that the transcripts tr_0 and tr_0' and the unauthenticated keys k_0 and k_0' match. Independent runs of tag-based authentication-only protocols for each client's factor ensure that C was alive at least during that part of the protocol execution. This is because at least one of those factors must remain uncorrupted prior to the acceptance of the server and all sub-protocol transcripts are linked together in the key confirmation step. Since tr_0 and tr_0' are linked to the transcripts of all authentication-only sub-protocols the key confirmation step further guarantees that C was alive during the UKE session and, hence, the secrecy of unauthenticated keys k_0 and k_0' follows from KE-security of the UKE protocol. The secrecy of k_0 and k_0' carries over to the secrecy of the final session keys k_C and k_S due to the use of independent random oracles. The optional server authentication follows the same reasoning as client authentication using PkA sessions. This intuition is proven in Theorems 1 and 2.

Theorem 1. *Our (α,β,γ)-MFAKE protocol is* AKE- *and* CAuth-*secure, in the random oracle model, and*

$$\begin{aligned}
\mathsf{Adv}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}}(\kappa) &\leq \mathsf{Adv}^{\mathsf{UKE},\mathcal{B}}_{\mathrm{KE}}(\kappa) + \alpha\cdot\mathsf{Adv}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa) + \beta\cdot\mathsf{Succ}^{\mathsf{tPkA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)\\
&\quad + \sum_{i=1}^{\gamma}\mathsf{Adv}^{\mathsf{tBiA}_i,\mathcal{B}}_{\mathrm{CAuth}}(\kappa) + (q^2_{\mathcal{H}_T} + q(q_{\mathcal{H}_C}+q_{\mathcal{H}_K}))\cdot 2^{-\kappa}, \text{and}\\
\mathsf{Adv}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFA},\mathcal{A}}_{\mathrm{CAuth}}(\kappa) &= \mathsf{Adv}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}}(\kappa) - q(q_{\mathcal{H}_k}-1)\cdot 2^{-\kappa}.
\end{aligned}$$

Proof. We prove this theorem using a series of games that are written for the AKE-security. To the end of the proof we discuss the impact of game hops on the CAuth-security. We denote by $\mathsf{Succ}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}\text{-}x}(\kappa)$ the success probability of $\mathcal{A}$ in game G_x and define

$$\varDelta_x(\kappa) = |\mathsf{Succ}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}\text{-}x}(\kappa) - \mathsf{Succ}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}\text{-}(x-1)}(\kappa)|.$$

G_0 This is the original AKE-security game, where the simulator answers the queries of $\mathcal{A}$ on behalf of the instances according to the specification of (α,β,γ)-MFAKE.

G_1 In this game for all simulated server and client instances that have matching UKE transcripts $\mathsf{tr}_0 = \mathsf{tr}_0'$ the corresponding UKE keys k_0 and k_0' are chosen at random such that $k_0 = k_0'$ holds. Otherwise, k_0 and k_0' are computed as in G_0.

Claim. $\varDelta_1(\kappa) \leq \mathsf{Adv}^{\mathsf{UKE},\mathcal{B}}_{\mathrm{KE}}(\kappa)$. *Proof.* For session instances that do not share matching UKE transcripts both games are identical. Any $\mathcal{A}$ that can distinguish between G_1 and G_0 with non-negligible probability can be used to break the KE security from Definition 4. The corresponding KE-adversary $\mathcal{B}$ against UKE would interact with $\mathcal{A}$ and simulate all its (α,β,γ)-MFAKE oracle queries as specified in G_0, except for the messages and keys of the UKE sub-protocol. Assume $\mathcal{A}$ invokes an instance of $\mathrm{U} \in \{C, S\}$. If this instance is supposed to send the first message in the UKE session then $\mathcal{B}$ queries its $\mathsf{Transcript}$ oracle and uses the first message of the obtained transcript as a response to $\mathcal{A}$. If $\mathcal{A}$ invokes an instance for $\mathrm{U}' \neq \mathrm{U}$ that is expected to send a message only after having received some incoming message then $\mathcal{B}$ waits for the corresponding Send query of $\mathcal{A}$ and checks whether input message is amongst those output by $\mathcal{B}$ from some transcript that it holds and responds with the next response message from this transcript. If the input message is unexpected then $\mathcal{B}$ runs UKE part on behalf of this instance of U' without consulting its oracle (and will thus be able to compute the UKE key for that session). Once UKE session on behalf of some instance is finished $\mathcal{B}$ has always a key to continue its simulation, either from its own UKE run or from a $\mathsf{Transcript}$ query. The way in which $\mathcal{B}$ simulates UKE sessions ensures that the latter type of keys are used in sessions that involve instances with matching UKE transcripts. If $\mathsf{Transcript}$ returns real keys then we are in G_0; otherwise in G_1. Hence, $\Delta_1(\kappa) \leq \mathsf{Adv}^{\mathsf{UKE},\mathcal{B}}_{\mathrm{KE}}(\kappa)$.

G_2 In this game the simulator aborts if in the ith tPwA session, for some $i \in \{1, \ldots, \alpha\}$, a server instance $[S, s']$ with tag t_S and (partial) transcript tr'_i accepts client C but there exists no instance of C with matching (partial) transcript tr_i and $t_C = t_S$, and $\boldsymbol{pwd}[i]$ is not corrupted.

Claim. $\Delta_2(\kappa) \leq \alpha \cdot \mathsf{Succ}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. *Proof.* We prove this with a hybrid argument using sub-games $\mathsf{G}_2^{\mathsf{upto}(j)}$, $j = 0, \ldots, \alpha$. Let tPwA_i, $i \in \{1, \ldots, \alpha\}$ denote the ith tPwA sub-protocol run. In $\mathsf{G}_2^{\mathsf{upto}(j)}$ all tPwA_i, $1 \leq i \leq j$ are handled as in G_2 and all tPwA_i, $j < i \leq \alpha$ are handled as in G_1. That is, $\mathsf{G}_1 = \mathsf{G}_2^{\mathsf{upto}(0)}$ and $\mathsf{G}_2 = \mathsf{G}_2^{\mathsf{upto}(\alpha)}$. As before, we define $\Delta_2^{\mathsf{upto}(j)}(\kappa)$ as the difference in $\mathcal{A}$'s success probability in two consecutive games $\mathsf{G}_2^{\mathsf{upto}(j-1)}$ and $\mathsf{G}_2^{\mathsf{upto}(j)}$. The difference between the two is that $\mathsf{G}_2^{\mathsf{upto}(j)}$ may still abort even if $\mathsf{G}_2^{\mathsf{upto}(j-1)}$ does not. Any $\mathcal{A}$ that can distinguish between the games must have successfully caused tPwA_j to abort in $\mathsf{G}_2^{\mathsf{upto}(j)}$, in which case an instance $[S, s']$ accepts C in tPwA_j while no partnered client instance with the same tag exists and no $\mathsf{CorruptClient}(C, 0, j)$ was asked. Such $\mathcal{A}$ can be used to break CAuth-security of tPwA. The simulator can act as CAuth-adversary $\mathcal{B}$ against tPwA by invoking new instances of the server in the tPwA game using tags of server instances that it simulates in the interaction with $\mathcal{A}$. The simulator relays all tPwA_j related queries of $\mathcal{A}$ as its own queries in the tPwA game and wins if $\mathcal{A}$ causes $\mathsf{G}_2^{\mathsf{upto}(j)}$ to abort. Therefore $\Delta_2^{\mathsf{upto}(j)}(\kappa) \leq \mathsf{Succ}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$ and thus $\Delta_2(\kappa) = \sum_{j=1}^{\alpha} \Delta_2^{\mathsf{upto}(j)}(\kappa) \leq \alpha \cdot \mathsf{Succ}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$.

G_3 In this game the simulator aborts if in the ith client side tPkA session, for some $i \in \{1, \ldots, \beta\}$, a server instance $[S, s']$ with tag t_S and (partial) transcript $\mathsf{tr}'_{\alpha+i}$ accepts client C but there exists no instance of C with matching (partial) transcript $\mathsf{tr}_{\alpha+i}$ and $t_C = t_S$, and $\boldsymbol{sk}_C[i]$ is not corrupted.

Claim. $\Delta_3(\kappa) \leq \beta \cdot \mathsf{Succ}^{\mathsf{tPkA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. *Proof.* We can use essentially the same hybrid argument as in G_2, but for tPkA sessions, and thus build a sequence of β sub-games to show that the difference between any two consecutive sub-games can be upper-bounded by $\mathsf{Succ}^{\mathsf{tPkA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. This leads to $\Delta_3(\kappa) = \sum_{j=1}^{\beta} \Delta_3^{\mathsf{upto}(j)}(\kappa) \leq \beta \cdot \mathsf{Succ}^{\mathsf{tPkA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$.

G_4 In this game the simulator aborts if in the ith tBiA session, for some $i \in \{1, \ldots, \gamma\}$, a server instance $[S, s']$ with tag t_S and (partial) transcript $\mathsf{tr}'_{\alpha+\beta+i}$ accepts client C but there exists no instance of C with matching (partial) transcript $\mathsf{tr}_{\alpha+\beta+i}$ and $t_C = t_S$, and the ith biometric is not corrupted.

Claim. $\Delta_4(\kappa) \leq \sum_{i=1}^{\gamma} \mathsf{Succ}^{\mathsf{tBiA}_i,\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. *Proof.* We denote by tBiA_i the tBiA protocol operating on the ith biometric. Again, using the hybrid argument as in G_2, but for tBiA sessions, we can build a sequence of γ sub-games and upper-bound the difference between any two consecutive sub-games $\mathsf{G}_4^{\mathsf{upto}(j-1)}$ and $\mathsf{G}_4^{\mathsf{upto}(j)}$ with $\mathsf{Succ}^{\mathsf{tBiA}_j,\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. The simulator can relay all tBiA_j related queries of $\mathcal{A}$ as its own queries in the tBiA game, including those related to the $\mathsf{BioComp}$ oracle since all biometric-dependent tBiA messages used in the (α, β, γ)-MFAKE protocol remain identical to those of the tBiA protocol. This leads to $\Delta_4(\kappa) = \sum_{j=1}^{\gamma} \Delta_4^{\mathsf{upto}(j)}(\kappa) \leq \sum_{i=1}^{\gamma} \mathsf{Succ}^{\mathsf{tBiA}_i,\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$.

Remark 5. If the simulation does not abort in this game then it is guaranteed that for each server instance $[S, s']$ that is entering the confirmation round with partial transcripts $\{\mathsf{tr}'_i\}_{1\le i\le\alpha+\beta+\gamma}$ (comprising executions of tPwA, PkA, and tBiA sub-protocols) and tag t_S, and that has not disqualified itself as a candidate for a Test query (i.e. fulfills freshness conditions from Section 2.2), there exists a client instance $[C, s]$ with partial transcripts $\{\mathsf{tr}_i\}_{1\le i\le\alpha+\beta+\gamma}$ such that there exists an index $i, 1 \le i \le \alpha+\beta+\gamma$ with $\mathsf{tr}_i = \mathsf{tr}'_i$. Moreover, any such client instance holds tag $t_C = t_S$.

G_5 In this game the simulation aborts if an instance $[S, s']$ enters the confirmation round with partial transcripts tr'_0 and $\{\mathsf{tr}'_i\}_{1\le i\le\alpha+\beta+\gamma}$ and there exists $[C, s]$ with partial transcripts tr_0 and $\{\mathsf{tr}_i\}_{1\le i\le\alpha+\beta+\gamma}$ such that for some index $i : \mathsf{tr}_i = \mathsf{tr}'_i$ but $\mathsf{tr}_0 \ne \mathsf{tr}'_0$.

Claim. $\varDelta_5(\kappa) \le q^2_{\mathcal{H}_T} 2^{-\kappa}$. *Proof.* G_4 already ensures that if $[S, s']$ accepts in all authentication sub-protocols then there exists a client instance with $t_C = t_S$. The only difference between the two games is that G_5 may still abort even if G_4 does not. If $\mathcal{A}$ can distinguish between the games then $\mathcal{A}$ must have successfully caused the simulator to abort in G_5, in which case $[S, s']$ and $[C, s]$ hold tags $t_S = t_C$ but $\mathsf{tr}_0 \ne \mathsf{tr}'_0$. We can thus output a collision for $\mathcal{H}_T$. Since $\mathcal{H}_T$ is a random oracle we get $\varDelta_5(\kappa) \le q^2_{\mathcal{H}_T} 2^{-\kappa}$.

Remark 6. G_5 implies that any instance $[S, s']$ that was not disqualified as a candidate for a Test query (upon entering the confirmation round) has a corresponding client instance $[C, s]$ with the same UKE transcript and at least one matching tag-based sub-protocol transcript.

G_6 This game proceeds as G_5, except that on behalf of an instance $[S, s']$ that is not disqualified as a candidate for a Test query the simulator computes $\mu_S \leftarrow \mathcal{H}'_C(s')$ and $k_S \leftarrow \mathcal{H}'_K(s')$ using two private random oracles $\mathcal{H}'_C$ and $\mathcal{H}'_k$, and sets $\mu_C = \mu_S$ and $k_C = k_S$ for the corresponding $[C, s]$ that has matching UKE transcript and at least one matching tag-based sub-protocol transcript.

Claim. $\varDelta_6(\kappa) \le q(q_{\mathcal{H}_C} + q_{\mathcal{H}_K}) \cdot 2^{-\kappa}$. Considering that in the previous game, confirmation values and session keys of $[S, s']$ were derived through random oracles $\mathcal{H}_C$ and $\mathcal{H}_k$ on input k'_0 (which is random as ensured by G_1) and the transcript s', any $\mathcal{A}$ that can distinguish between the games must ask at some point a query for $\mathcal{H}_C$ or $\mathcal{H}_k$ containing k'_0 and s' for any of the q invoked sessions as input. Therefore, $\varDelta_6(\kappa) \le q(q_{\mathcal{H}_C} + q_{\mathcal{H}_k}) \cdot 2^{-\kappa}$.

G_6 implies that if $[S, s']$ accepts and is not disqualified as a candidate for a Test query then k_S is uniformly distributed in the domain of session keys. Hence, the probability of $\mathcal{A}$ to win in G_6 no longer depends on the key, i.e. $\mathcal{A}$ can win in G_6 only by guessing bit b (with probability $\frac{1}{2}$).

Summarizing the probability differences across all games we obtain

$$\begin{aligned}\mathsf{Adv}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}}(\kappa) &= \left|\mathsf{Succ}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}_{\mathrm{AKE}}(\kappa) - q\left(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma}\mathsf{false}^{\mathrm{pos}}_i\right) - \frac{1}{2}\right| \\ &= \left|\sum_{i=1}^{6}\varDelta_i(\kappa) + \frac{1}{2} - q\left(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma}\mathsf{false}^{\mathrm{pos}}_i\right) - \frac{1}{2}\right|.\end{aligned}$$

Taking into account that

$$\begin{aligned}\sum_{i=1}^{6}\varDelta_i(\kappa) \le \mathsf{Adv}^{\mathsf{UKE},\mathcal{B}}_{\mathrm{KE}}(\kappa) + \alpha\cdot\mathsf{Succ}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa) + \beta\cdot\mathsf{Succ}^{\mathsf{tPkA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa) \\ + \sum_{i=1}^{\gamma}\mathsf{Succ}^{\mathsf{tBiA}_i,\mathcal{B}}_{\mathrm{CAuth}}(\kappa) + (q^2_{\mathcal{H}_T} + q(q_{\mathcal{H}_C} + q_{\mathcal{H}_K}))\cdot 2^{-\kappa},\end{aligned}$$

and that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa) &= \left|\mathsf{Succ}^{\mathsf{tPwA},\mathcal{B}}_{\mathrm{CAuth}}(\kappa) - \frac{q}{|\mathcal{D}_{\mathtt{pwd}}|}\right| \\ \mathsf{Adv}^{\mathsf{tBiA}_i,\mathcal{B}}_{\mathrm{CAuth}}(\kappa) &= \left|\mathsf{Succ}^{\mathsf{tBiA}_i,\mathcal{B}}_{\mathrm{CAuth}}(\kappa) - q\cdot\mathsf{false}^{\mathrm{pos}}_i\right|\end{aligned}$$

we obtain

$$\begin{aligned}\mathsf{Adv}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) \leq \mathsf{Adv}_{\mathrm{KE}}^{\mathsf{UKE},\mathcal{B}}(\kappa) + \alpha \cdot \mathsf{Adv}_{\mathrm{CAuth}}^{\mathsf{tPwA},\mathcal{B}}(\kappa) + \beta \cdot \mathsf{Succ}_{\mathrm{CAuth}}^{\mathsf{tPkA},\mathcal{B}}(\kappa) \\ + \sum_{i=1}^{\gamma} \mathsf{Adv}_{\mathrm{CAuth}}^{\mathsf{tBiA}_i,\mathcal{B}}(\kappa) + (q_{\mathcal{H}_T}^2 + q(q_{\mathcal{H}_C} + q_{\mathcal{H}_K})) \cdot 2^{-\kappa},\end{aligned}$$

which is negligible by assumptions on UKE, tPwA, tPkA, and tBiA.

Proof for CAuth*-security.* With regard to client authentication, consider the above game sequence from the perspective of the CAuth-security game and success probability $\mathsf{Succ}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa)$. Freshness conditions regarding server instances encompass the requirements that are relevant for the CAuth-game. Then, Remark 6 implies that in G_5 for each server instance $[S, s']$ for which $\mathcal{A}$ could still win the game there exists a client instance $[C, s]$ with the matching UKE transcript and at least one matching tag-based sub-protocol transcript. In G_6, μ_C and μ_S are computed using private oracle, while for CAuth-security modifications of k_C and k_S are irrelevant. The probability difference to G_5 is thus upper-bounded by $q \cdot q_{\mathcal{H}_C} \cdot 2^{-\kappa}$. Then, $[S, s']$ must have received $\mu_C = \mu_S$ without having a partnered client instance. That is $\mathcal{A}$ must have asked a Send query containing a value that matches a uniformly distributed μ_S. This happens with probability at most $q \cdot 2^{-\kappa}$ for up to q invoked server instances. We thus obtain the following CAuth-success

$$\mathsf{Succ}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) = \mathsf{Succ}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q(q_{\mathcal{H}_k} - 1) \cdot 2^{-\kappa} - \frac{1}{2}.$$

Taking into account that by definition

$$\mathsf{Adv}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) = \left| \mathsf{Succ}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q\left(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{pos}}\right) - \frac{1}{2} \right|$$

we obtain a negligible CAuth-advantage

$$\begin{aligned}\mathsf{Adv}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) &= \left| \mathsf{Succ}_{\mathrm{CAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q\left(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{pos}}\right) \right| \\ &= \left| \mathsf{Succ}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q(q_{\mathcal{H}_k} - 1) \cdot 2^{-\kappa} - \frac{1}{2} \right. \\ &\qquad \left. - q\left(\frac{\alpha}{|\mathcal{D}_{\mathtt{pwd}}|} + \sum_{i=1}^{\gamma} \mathsf{false}_i^{\mathrm{pos}}\right) \right| \\ &= \left| \mathsf{Adv}_{\mathrm{AKE}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - q(q_{\mathcal{H}_k} - 1) \cdot 2^{-\kappa} \right|.\end{aligned}$$

□

Theorem 2. *Our* (α, β, γ)-MFAKE *protocol with server authentication is* SAuth*-secure in the random oracle model, and*

$$\mathsf{Succ}_{\mathrm{SAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) \leq \mathsf{Adv}_{\mathrm{KE}}^{\mathsf{UKE},\mathcal{B}}(\kappa) + \mathsf{Succ}_{\mathrm{CAuth}}^{\mathsf{tPkA},\mathcal{B}}(\kappa) + (q_{\mathcal{H}_T}^2 + q(q_{\mathcal{H}_S} + 1)) \cdot 2^{-\kappa}.$$

Proof. This proof resembles in part the proof of Theorem 1 and proceeds in a series of similar games. We denote by $\mathsf{Succ}_{\mathrm{SAuth}\text{-}x}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa)$ the success probability of $\mathcal{A}$ in game G_{x}. For each game G_{x}, we define $\varDelta_x(\kappa)$ as the difference in $\mathcal{A}$'s success probability when playing against the two consecutive games $\mathsf{G}_{\mathrm{x\text{-}1}}$ and G_{x}, i.e., $\varDelta_x(\kappa) = |\mathsf{Succ}_{\mathrm{SAuth}\text{-}x}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) - \mathsf{Succ}_{\mathrm{SAuth}\text{-}(x-1)}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa)|$.

G_0 This is the original SAuth-security game, where the simulator answers the queries of $\mathcal{A}$ on behalf of the instances according to the specification of (α, β, γ)-MFAKE.

G_1 This game proceeds as G_0, except that for all simulated server and client instances that have matching UKE transcripts $\mathsf{tr}_0 = \mathsf{tr}'_0$ the corresponding UKE keys k_0 and k'_0 are chosen at random such that $k_0 = k'_0$ holds. Otherwise, k_0 and k'_0 are computed as in G_0.

Claim. $\Delta_1(\kappa) \leq \mathsf{Adv}_{\mathrm{KE}}^{\mathsf{UKE},\mathcal{B}}(\kappa)$. *Proof.* For client and server instances that do not share matching UKE transcripts both games are identical. Any $\mathcal{A}$ that can distinguish between G_1 and G_0 with non-negligible probability can be used to break the KE security from Definition 4. The description of the UKE adversary is exactly the same as in G_1 from the proof of Theorem 1. Hence, $\Delta_1(\kappa) \leq \mathsf{Adv}_{\mathrm{KE}}^{\mathsf{UKE},\mathcal{B}}(\kappa)$, as claimed.

G_2 This game proceeds as G_1, except that the simulator aborts if in the server-side tPkA session a client instance $[C, s]$ with tag t_C and (partial) transcript $\mathsf{tr}_{\alpha+\beta+\gamma+1}$ accepts server S but there exists no instance of S with matching (partial) transcript $\mathsf{tr}'_{\alpha+\beta+\gamma+1}$ and tag $t_S = t_C$, and sk_S is not corrupted.

Claim. $\Delta_2(\kappa) \leq \mathsf{Succ}_{\mathrm{CAuth}}^{\mathsf{tPkA},\mathcal{B}}(\kappa)$. *Proof.* As already described in games G_2 through G_4 in proof of Theorem 1 if $\mathcal{A}$ can distinguish between the two games, it can be immediately used to break CAuth-security of tPkA. (In this game CAuth-security is understood as a security property of PkA in case where the authenticating party is the server S. Recall that PkA offers single-side authentication and was defined from the perspective of an authenticating client. In this game the authenticating party is S but the notion of CAuth-security remains as defined.) Hence, $\Delta_2(\kappa) \leq \mathsf{Succ}_{\mathrm{CAuth}}^{\mathsf{tPkA},\mathcal{B}}(\kappa)$, as claimed.

Remark 7. Note that if the simulation does not abort in G_2 then it is guaranteed that for each client instance $[C, s]$ that is entering the confirmation round with partial transcript $\mathsf{tr}_{\alpha+\beta+\gamma+1}$ and tag t_C, there exists a server instance $[S, s']$ with partial transcript $\mathsf{tr}'_{\alpha+\beta+\gamma+1} = \mathsf{tr}_{\alpha+\beta+\gamma+1}$ and $t_S = t_C$ if neither C nor $S = \mathsf{pid}([C, s])$ has been fully corrupted.

G_3 This game proceeds as G_2, except that simulation aborts if an instance $[C, s]$ enters the confirmation round with partial transcripts tr_0 and $\mathsf{tr}_{\alpha+\beta+\gamma+1}$ and there exists $[S, s']$ with partial transcripts tr'_0 and $\mathsf{tr}'_{\alpha+\beta+\gamma+1}$ such that $\mathsf{tr}_{\alpha+\beta+\gamma+1} = \mathsf{tr}'_{\alpha+\beta+\gamma+1}$ but $\mathsf{tr}_0 \neq \mathsf{tr}'_0$.

Claim. $\Delta_3(\kappa) \leq q^2_{\mathcal{H}_T} 2^{-\kappa}$. *Proof.* G_2 already ensures that if $[C, s']$ accepts in the server side tPkA sub-protocol then there exists a server instance with $t_S = t_C$. The only difference between the two games is that G_3 may still abort even if G_2 does not. If $\mathcal{A}$ can distinguish between the games then $\mathcal{A}$ must have successfully caused the simulator to abort in G_3, in which case $[C, s]$ and $[S, s']$ hold tags $t_C = t_S$ but $\mathsf{tr}_0 \neq \mathsf{tr}'_0$. We can thus output a collision for $\mathcal{H}_T$. Since $\mathcal{H}_T$ is a random oracle we get $\Delta_3(\kappa) \leq q^2_{\mathcal{H}_T} 2^{-\kappa}$, as claimed.

G_4 This game proceeds as G_3, except that on behalf of an instance $[C, s]$ for which neither C nor $S = \mathsf{pid}([C, s])$ is fully corrupted the simulator computes $\nu_C \leftarrow \mathcal{H}'_S(s')$ using a private random oracle $\mathcal{H}'_S$, and sets $\nu_S = \nu_C$ for the corresponding $[S, s']$ that has matching UKE transcript and matching server-side tPkA sub-protocol transcript.

Claim. $\Delta_4(\kappa) \leq q \cdot q_{\mathcal{H}_S} \cdot 2^{-\kappa}$. *Proof.* Considering that in the previous game, confirmation values of $[C, s]$ were derived through the random oracle $\mathcal{H}_S$ on input k_0 (which is random as ensured by G_1) and the transcript s, any $\mathcal{A}$ that can distinguish between the games must ask at some point a query for $\mathcal{H}_S$ containing k_0 and s for any of the q invoked sessions as input. Therefore, $\Delta_4(\kappa) \leq q \cdot q_{\mathcal{H}_S} \cdot 2^{-\kappa}$, as claimed.

Assume that $\mathcal{A}$ wins in G_4. Then, $[C, s]$ must have received $\nu_S = \nu_C$ without having a partnered server instance. That is, $\mathcal{A}$ must have asked a Send query containing a value that matches a uniformly distributed ν_C. This happens with probability at most $q \cdot 2^{-\kappa}$ for up to q invoked client instances. We thus get

$$\begin{aligned}\mathsf{Succ}_{\mathrm{SAuth}}^{(\alpha,\beta,\gamma)\text{-}\mathsf{MFAKE},\mathcal{A}}(\kappa) &= \sum_{i=1}^{4} \Delta_i(\kappa) + q \cdot 2^{-\kappa} \\ &\leq \mathsf{Adv}_{\mathrm{KE}}^{\mathsf{UKE},\mathcal{B}}(\kappa) + \mathsf{Succ}_{\mathrm{CAuth}}^{\mathsf{tPkA},\mathcal{B}}(\kappa) + (q^2_{\mathcal{H}_T} + q(q_{\mathcal{H}_S} + 1)) \cdot 2^{-\kappa},\end{aligned}$$

which is negligible by assumptions on UKE and tPkA.

□

4 Conclusion

The proposed framework for multi-factor authentication and key exchange protocols enables black-box constructions from existing, better-understood single-factor authentication-only schemes. Our generic construction of the (α, β, γ)-MFAKE protocol avoids undesirable interactions amongst the different factors and bears optimization potential since messages of tag-based authentication sub-protocols can be interleaved or communicated over different channels. Thanks to its modularity the framework can easily be extended in the future to accommodate other authentication factors, e.g. based on friend-of-friend or social authentication [13,17].

Acknowledgments

Nils Fleischhacker was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy, and Accountability (CISPA, `www.cispa-security.org`). Mark Manulis was supported by the German Research Foundation (DFG), project PRIMAKE (MA 4957).

References

1. PCI Data Security Standard, Ver.2. `http://www.pcisecuritystandards.org/`, 2010.
2. NIST Special Publication 800-63, Rev.1. `http://csrc.nist.gov/publications/`, 2011.
3. M. Abdalla, O. Chevassut, and D. Pointcheval. One-Time Verifier-Based Encrypted Key Exchange. In *PKC 2005*, volume 3386 of *LNCS*, pages 47–64. Springer, 2005.
4. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In *Public Key Cryptography (PKC 2005)*, volume 3386 of *LNCS*, pages 65–84. Springer, 2005.
5. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, 2005.
6. M. Abdalla and D. Pointcheval. Simple Password-Based Encrypted Key Exchange Protocols. In *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, 2005.
7. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.
8. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO 1993*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.
9. F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, Report 2013/833, 2013. `http://eprint.iacr.org/2013/833`.
10. S. Blake-Wilson, D. Johnson, and A. Menezes. Key Agreement Protocols and Their Security Analysis. In *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *LNCS*, pages 30–45. Springer, 1997.
11. X. Boyen. Reusable Cryptographic Fuzzy Extractors. In *ACM CCS 2004*, pages 82–91. ACM, 2004.
12. X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith. Secure Remote Authentication Using Biometric Data. In *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 147–163. Springer, 2005.
13. J. G. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-Factor Authentication: Somebody You Know. In *ACM CCS 2006*, pages 168–178. ACM, 2006.
14. J. Bringer and H. Chabanne. An Authentication Protocol with Encrypted Biometric Data. In *AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 109–124. Springer, 2008.
15. J. Bringer, H. Chabanne, M. Izabachène, D. Pointcheval, Q. Tang, and S. Zimmer. An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication. In *ACISP 2007*, volume 4586 of *LNCS*, pages 96–106. Springer, 2007.
16. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.
17. E. De Cristofaro, M. Manulis, and B. Poettering. Private Discovery of Common Social Contacts. *International Journal of Information Security*, 12(1):49–65, 2013.
18. Y. Dodis, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, 2004.
19. T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In *CRYPTO 2008*, volume 5157 of *LNCS*, pages 203–220. Springer, 2008.

20. Federal Financial Institutions Examination Council. Authentication in an Internet Banking Environment, 2005. `http://www.ffiec.gov/pdf/authentication_guidance.pdf`.
21. F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In *ESORICS 2008*, volume 5283 of *LNCS*, pages 97–114. Springer, 2008.
22. F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In *ACM CCS 2010*, pages 250–259. ACM, 2010.
23. C. Gentry, P. D. MacKenzie, and Z. Ramzan. Password authenticated key exchange using hidden smooth subgroups. In *ACM CCS 2005*, pages 299–309. ACM, 2005.
24. C. Gentry, P. D. MacKenzie, and Z. Ramzan. A Method for Making Password-Based Key Exchange Resilient to Server Compromise. In *CRYPTO 2006*, volume 4117 of *LNCS*, pages 142–159. Springer, 2006.
25. F. Hao. On Robust Key Agreement Based on Public Key Authentication. In *Financial Cryptography and Data Security*, volume 6052 of *LNCS*, pages 383–390. Springer, 2010.
26. F. Hao and D. Clarke. Security Analysis of a Multi-factor Authenticated Key Exchange Protocol. In *ACNS*, volume 7341 of *LNCS*, pages 1–11. Springer, 2012.
27. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. Generic Compilers for Authenticated Key Exchange. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 232–249. Springer, 2010.
28. J. Katz, R. Ostrovsky, and M. Yung. Forward Secrecy in Password-Only Key Exchange Protocols. In *SCN 2002*, volume 2576 of *LNCS*, pages 29–44. Springer, 2002.
29. F. Kiefer and M. Manulis. Zero-Knowledge Password Policy Checks and Verifier-Based PAKE. In *ESORICS 2014*, volume 8713 of *LNCS*, pages 295–312. Springer, 2014.
30. B. A. LaMacchia, K. Lauter, and A. Mityagin. Stronger Security of Authenticated Key Exchange. In *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16, 2007.
31. Y. Lee, S. Kim, and D. Won. Enhancement of Two-Factor Authenticated Key Exchange Protocols in Public Wireless LANs. *Computers & Electrical Engineering*, 36(1):213–223, 2010.
32. C.-T. Li and M.-S. Hwang. An Efficient Biometrics-Based Remote User Authentication Scheme Using Smart Cards. *Journal of Network and Computer Applications*, 33(1):1–5, 2010.
33. X. Li, J.-W. Niu, J. Ma, W.-D. Wang, and C.-L. Liu. Cryptanalysis and Improvement of a Biometrics-Based Remote User Authentication Scheme Using Smart Cards. *Journal of Network and Computer Applications*, 34(1):73–79, 2011.
34. Y. M. Park and S. K. Park. Two Factor Authenticated Key Exchange (TAKE) Protocol in Public Wireless LANs. *IEICE Transactions on Communications*, E87-B(5):1382–1385, 2004.
35. K. G. Paterson and D. Stebila. One-Time-Password-Authenticated Key Exchange. In *ACISP 2010*, volume 6168 of *LNCS*, pages 264–281. Springer, 2010.
36. D. Pointcheval and S. Zimmer. Multi-factor Authenticated Key Exchange. In *ACNS 2008*, volume 5037 of *LNCS*, pages 277–295. Springer, 2008.
37. R. Song. Advanced Smart Card Based Password Authentication Protocol. *Computer Standards & Interfaces*, 32(5–6):321–325, 2010.
38. D. Stebila, P. Udupi, and S. Chang. Multi-Factor Password-Authenticated Key Exchange. In *Eighth Australasian Information Security Conference (AISC 2010)*, volume 105, pages 56–66, 2010.
39. J. E. Tapiador, J. C. Hernandez-Castro, P. Peris-Lopez, and J. A. Clark. Cryptanalysis of Song's Advanced Smart Card Based Password Authentication Protocol. arXiv.org, Cryptography and Security, 2011. `http://arxiv.org/abs/1111.2744v1`.
40. X. Wang and W. Zhang. An Efficient and Secure Biometric Remote User Authentication Scheme Using Smart Cards. In *Pacific-Asia Workshop on Computational Intelligence and Industrial Application (PACIIA '08)*, volume 2, pages 913–917. IEEE, 2008.

A Some Special Cases of (α, β, γ)-MFAKE

This section relates our (α, β, γ)-MFAKE model to some well-known authentication models. In particular, proven relationship between $(1, 0, 0)$-MFAKE and the PAKE model from [7] implies that any PAKE protocol proven secure in [7] can readily be used as an $(1, 0, 0)$-MFAKE in our generic solution; similarly, any 2-AKE protocol that satisfies the security definitions from [8], refined in [10] for the public-key setting, can readily be used as an $(0, 1, 0)$-MFAKE.

Special Case of $(1, 0, 0)$**-MFAKE** Using $(\alpha, \beta, \gamma) = (1, 0, 0)$ we obtain the original setting of Password-based Authenticated Key Exchange (PAKE) as defined in [7]. This relationship is established in Theorem 3.

Theorem 3. *(1) Let $\mathcal{P}$ be an AKE-secure PAKE protocol with client authentication according to the definitions by Bellare, Pointcheval, and Rogaway [7], then $\mathcal{P}$ is also an AKE-secure $(1,0,0)$-MFAKE protocol with client authentication. (2) Let $\mathcal{P}$ be an AKE-secure $(1,0,0)$-MFAKE protocol with client authentication. If $\mathcal{P}$ is symmetric, i.e. the roles of server and client can be swapped, then $\mathcal{P}$ is also an AKE-secure PAKE protocol with client and server authentication as defined in [7].*

Proof. We denote the notions of AKE-security, client authentication, and server authentication from [7] by AKE_{BPR}, CAuth_{BPR}, and SAuth_{BPR}, respectively. For corresponding definitions we refer to [7]. The two statements in the theorem are proven separately.

Statement (1) Let $\mathcal{A}^{\text{AKE}}$ be an adversary against the AKE-security of $\mathcal{P}$ and $\mathcal{A}^{\text{CAuth}}$ an adversary against the CAuth-security of $\mathcal{P}$ in the $(1,0,0)$-MFAKE model. We use $\mathcal{A}^{\text{AKE}}$ to construct an adversary $\mathcal{B}^{\text{AKE}_{BPR}}$ against the AKE_{BPR}-security of $\mathcal{P}$ and $\mathcal{A}^{\text{CAuth}}$ to construct $\mathcal{B}^{\text{CAuth}_{BPR}}$ against the client authentication of $\mathcal{P}$ in the model from [7].

Simulation of Oracles: For simplicity we write $\mathcal{A}$ for an adversary in our $(1,0,0)$-MFAKE model and $\mathcal{B}$ for an adversary (against the same security property as $\mathcal{A}$) in the model from [7]. The general idea is that $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine and simulates its oracles in the $(1,0,0)$-MFAKE model using the oracles of $\mathcal{B}$ from [7]. Note that oracles $\mathsf{BioComp}$ and $\mathsf{CorruptServer}$ of $\mathcal{A}$ need not to be handled, as the theorem views $\mathcal{P}$ as a PAKE protocol that (by definition) does not use biometrics or public keys.

If $\mathcal{A}$ invokes a new session between C and S via $\mathsf{Invoke}(C,S)$ then $\mathcal{B}$ queries $\mathsf{Send}(C,s,S)$ with a previously unused s and returns its output to $\mathcal{A}$. (In [7] protocol sessions are invoked through this special Send query.) Queries of $\mathcal{A}$ to Send, $\mathsf{RevealSK}$, and $\mathsf{CorruptClient}$ oracles are answered by $\mathcal{B}$ as follows: If $\mathcal{A}$ queries $\mathsf{Send}([\text{U},s],m)$ with $\text{U} \in \{C,S\}$ then $\mathcal{B}$ queries $\mathsf{Send}(\text{U},s,m)$ and returns its output to $\mathcal{A}$. If $\mathcal{A}$ queries $\mathsf{RevealSK}([\text{U},s])$ with $\text{U} \in \{C,S\}$ then $\mathcal{B}$ returns the output of its own $\mathsf{Reveal}(\text{U},s)$ query from [7]. In $(1,0,0)$-MFAKE queries to $\mathsf{CorruptClient}$ of $\mathcal{A}$ can only be of the form $\mathsf{CorruptClient}(C,1,1)$. Any such query is forwarded by $\mathcal{B}$ as $\mathsf{Corrupt}(C, \textsc{DontChange})$ from [7] and the response is forwarded to $\mathcal{A}$. (Note that in [7], a special symbol $\textsc{DontChange}$ means that upon corruption of the client the latter's password remains unchanged on the server's side.) In this way $\mathcal{B}$ can perfectly simulate Invoke, Send, $\mathsf{RevealSK}$, and $\mathsf{CorruptClient}$ queries of $\mathcal{A}$.

AKE-Security: Let $\mathcal{A} = \mathcal{A}^{\text{AKE}}$ and $\mathcal{B} = \mathcal{B}^{\text{AKE}_{BPR}}$. Here $\mathcal{B}$ needs to simulate the additional Test oracle of $\mathcal{A}$ that can be queried for any fresh server instances. $\mathcal{B}$ also has access to a Test oracle in the model from [7]. The difference is that [7] uses Find-then-Guess (FtG) approach, where only one Test query can be asked, whereas $(1,0,0)$-MFAKE follows the Real-or-Random (RoR) approach with multiple Test queries. The difference between RoR and FtG was explored in [5, Lemma 2], resulting in $\mathsf{Adv}^{\mathcal{P},\mathcal{B}_{RoR}}_{\text{AKE}_{BPR}}(\kappa) \le q_{\mathsf{Test}} \cdot \mathsf{Adv}^{\mathcal{P},\mathcal{B}_{FtG}}_{\text{AKE}_{BPR}}(\kappa)$, where q_{Test} is the maximum number of Test queries asked by an AKE-adversary $\mathcal{B}_{RoR}$ in the RoR version of [7]. We use this result as an intermediate step, i.e. we first consider $\mathcal{B} = \mathcal{B}^{\text{AKE}_{BPR}}_{RoR}$ within our reduction (that is as an AKE-adversary for the RoR version of [7]) and let $\mathcal{B}$ answer each $\mathsf{Test}([S,s])$ query of $\mathcal{A}$ by forwarding it as its own $\mathsf{Test}(S,s)$ query and returning the output back to $\mathcal{A}$. Once $\mathcal{A}$ outputs bit b', $\mathcal{B}$ forwards this bit as its own output in the AKE-security game from the RoR version of [7]. If $\mathcal{A}$ correctly outputs bit $b' = b$ used by the Test oracle, then so is $\mathcal{B}$. This is guaranteed by the definitions of freshness. Then by switching to $\mathcal{B} = \mathcal{B}^{\text{AKE}_{BPR}}_{FtG}$ and using [5, Lemma 2] we obtain $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\text{AKE}}(\kappa) \le q_{\mathsf{Test}} \cdot \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\text{AKE}_{BPR}}(\kappa)$, where q_{Test} is the number of Test queries asked by the AKE adversary $\mathcal{A} = \mathcal{A}^{\text{AKE}}$ in the $(1,0,0)$-MFAKE model.

Client Authentication: Let $\mathcal{A} = \mathcal{A}^{\text{CAuth}}$ and $\mathcal{B} = \mathcal{B}^{\text{CAuth}_{BPR}}$. This case is simpler since there are no Test queries to consider. If $\mathcal{A}$ is successful then at the end of the simulation there must exist a fresh instance of server S that has accepted without a partnered instance of client C. This means that $\mathcal{B}$ is successful whenever $\mathcal{A}$ is and thus $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\text{CAuth}}(\kappa) \le \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\text{CAuth}_{BPR}}(\kappa)$.

Statement (2) Let $\mathcal{A}^{\text{AKE}_{BPR}}$ be an AKE_{BPR}-adversary against $\mathcal{P}$, $\mathcal{A}^{\text{CAuth}_{BPR}}$ an adversary against client authentication of $\mathcal{P}$, and $\mathcal{A}^{\text{SAuth}_{BPR}}$ an adversary against server authentication of $\mathcal{P}$ in the model from [7]. We first use $\mathcal{A}^{\text{AKE}_{BPR}}$ to construct an AKE-adversary $\mathcal{B}^{\text{AKE}}$ against $(1,0,0)$-MFAKE. We then show that (since $\mathcal{P}$ is assumed to be symmetric) both $\mathcal{A}^{\mathrm{CAuth}_{BPR}}$ and $\mathcal{A}^{\mathrm{SAuth}_{BPR}}$ can be used to construct $\mathcal{B}^{\mathrm{CAuth}}$ against client authentication in $(1, 0, 0)$-MFAKE.

Simulation of Oracles: For simplicity we write $\mathcal{A}$ for an adversary from [7] and $\mathcal{B}$ for the corresponding adversary in the $(1, 0, 0)$-MFAKE model. As in Statement (1), $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine and simulates its oracles using the oracles for $(1, 0, 0)$-MFAKE. Since an $(1, 0, 0)$-MFAKE protocol $\mathcal{P}$ does not use biometrics and public keys $\mathcal{B}$ has no use for its $\mathsf{BioComp}$ and $\mathsf{CorruptServer}$ oracles. We discuss now, how $\mathcal{B}$ answers queries of $\mathcal{A}$ to its $\mathsf{Execute}$, Send, Reveal, and $\mathsf{Corrupt}$ oracles from [7]. An $\mathsf{Execute}(C, s, S, s')$ query of $\mathcal{A}$, which is supposed to return a complete transcript of an honest protocol execution between the two specified instances is answered by $\mathcal{B}$ as follows. $\mathcal{B}$ calls $\mathsf{Invoke}(C, S)$ to establish a new client instance $[C, s]$ with partner id S and $\mathsf{Invoke}(S, C)$ to establish a new server instance $[S, s']$ with partner id C. Then, using its Send oracle $\mathcal{B}$ faithfully forwards messages between the both instances until it obtains the resulting protocol transcript that it hands over to $\mathcal{A}$. The Send oracle in the model from [7] serves two purposes: it invokes new client and server instances and facilitates communication amongst them. In contrast in our $(1, 0, 0)$-MFAKE model sessions are invoked using the Invoke oracle. Hence, if $\mathcal{A}$ queries $\mathsf{Send}(C, s, S)$ from [7] to invoke a new client session with server S, $\mathcal{B}$ calls $\mathsf{Invoke}(C, S)$, returns its output to $\mathcal{A}$, and records the resulting instance as the s-th instance of C with server S as its communication partner. Similarly, if $\mathcal{A}$ queries $\mathsf{Send}(S, s', C)$ to invoke a new server session with client C, $\mathcal{B}$ calls $\mathsf{Invoke}(S, C)$, returns its output to $\mathcal{A}$, and records the resulting instance as the s'-th instance of S with client C as its communication partner. $\mathcal{A}$ controls communication amongst the instances using $\mathsf{Send}(\mathrm{U}, s, M)$, which sends message M to the s-th instance of $\mathrm{U} \in \{C, S\}$. These queries are forwarded by $\mathcal{B}$ as its own Send queries in the $(1, 0, 0)$-MFAKE model and their response is returned to $\mathcal{A}$. A $\mathsf{Reveal}(\mathrm{U}, s)$ query of $\mathcal{A}$ is answered by $\mathcal{B}$ through its own $\mathsf{RevealSK}([\mathrm{U}, s])$ query. A $\mathsf{Corrupt}(C, pw)$ query from [7] serves two purposes: first, the adversary gets the current password of the client C. In addition, if $pw \neq \textsc{DontChange}$, where $\textsc{DontChange}$ is considered as a special symbol, the current password for C stored at server S is replaced with pw that is provided as part of the query. Note that Katz, Ostrovsky, and Yung [28] later refined the $\mathsf{Corrupt}$ query from [7] by allowing the adversary to install a new password for C at S without necessarily learning the previous password. Our proof can cope with their refinement. In our simulation, any query $\mathsf{Corrupt}(C, \textsc{DontChange})$ of $\mathcal{A}$ is answered by $\mathcal{B}$ using $\mathsf{CorruptClient}(C, 1, 1)$ query. Whereas, if $\mathcal{A}$ asks $\mathsf{Corrupt}(C, pw)$ with $pw \neq \textsc{DontChange}$ (or wishes to replace the current password with pw as in [28]) then $\mathcal{B}$ records pw as a new password for C and answers subsequent Send queries of $\mathcal{A}$ to the new instances of S with partner id C, by executing the protocol on behalf of S with the new password pw. This simulation is possible since the $(1, 0, 0)$-MFAKE protocol $\mathcal{P}$ is assumed to be symmetric, so the roles of C and S can be swapped. It is thus sufficient for $\mathcal{B}$ to know the password shared between C and S in order to execute the protocol on behalf of an instance of S that was invoked with partner id C. Note that in [7] any instance of S with partner id C and any instance of C with partner id S becomes unfresh if $\mathcal{A}$ issues Send queries to these instances after having modified the password for C at S.

AKE-Security: Let $\mathcal{A} = \mathcal{A}^{\mathrm{AKE}_{BPR}}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{AKE}}$. In this case $\mathcal{B}$ must additionally simulate the Test oracle that is available to $\mathcal{A}$ in [7]. Note that also $\mathcal{B}$ has access to a Test oracle in the $(1, 0, 0)$-MFAKE model but there is a differences between the two models. The first difference (which was important for Statement (1) above) is that $\mathcal{B}$ may ask multiple Test queries, whereas $\mathcal{A}$ may issue only one Test query. This difference is not important for Statement (2), since $\mathcal{B}$ executes $\mathcal{A}$ as a subroutine and will thus ask only one Test query. The second difference, which is important for Case (2), is that $\mathcal{B}$ can query its Test oracle with respect to server instances only (recall that in the (α, β, γ)-MFAKE setting servers do not necessarily have secret keys), whereas $\mathcal{A}$ in the PAKE setting from [7] can direct its Test query also to a client instance. We deal with this difference in the following way. The universe of PAKE participants in the model from [7] is split into two sets, Client_{BPR} and Server_{BPR}. Also in the (α, β, γ)-MFAKE setting each participant is either a client C or a server S. We use the fact that in Statement (2) the $(1, 0, 0)$-MFAKE protocol $\mathcal{P}$ is assumed to be symmetric and let $\mathcal{B}$ first flip a bit $c \in_R \{0, 1\}$ aiming to guess with probability $\frac{1}{2}$ whether $\mathcal{A}$ will ask its Test query to a server instance ($c = 0$) or to a client instance ($c = 1$). Depending on the value of c, $\mathcal{B}$ defines the two sets of participants as follows: if $c = 0$ then $\mathcal{B}$ defines Client_{BPR} to be the set of all clients C and Server_{BPR} to contain all servers S, otherwise if $c = 1$ then Client_{BPR} is defined as the set of all servers S and Server_{BPR} as the set of all clients C. Once the two sets are defined, $\mathcal{B}$ invokes $\mathcal{A}$ and proceeds with simulation. If the guess of $\mathcal{B}$ is correct then $\mathcal{B}$ can always forward the $\mathsf{Test}(\mathrm{U}, s)$ query of $\mathcal{A}$ as its own $\mathsf{Test}([S, s])$ query where S is a server in the $(1, 0, 0)$-MFAKE model. Indeed, if $c = 0$ then $\mathrm{U} = S$ for some $S \in \mathit{Server}_{BPR}$, whereas if $c = 1$ then $\mathrm{U} = S$ for some $S \in \mathit{Client}_{BPR}$. In both cases $\mathcal{B}$ will forward the answer of its own Test query to $\mathcal{A}$. Once $\mathcal{A}$ outputs bit b', indicating the end of the AKE-security game from [7], $\mathcal{B}$ forwards this bit as its own output in the AKE-security game from the $(1, 0, 0)$-MFAKE model. If $\mathcal{A}$ is successful, i.e. correctly outputs bit $b' = b$ used by the Test oracle, then so is $\mathcal{B}$. This is guaranteed by the definitions of freshness in both models. This gives us the desired inequality $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{AKE}_{BPR}}(\kappa) \leq 2\mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{AKE}}(\kappa)$.

Client Authentication: Let $\mathcal{A} = \mathcal{A}^{\mathrm{CAuth}}_{BPR}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{CAuth}}$. In this case there are no additional oracles to take care of. If $\mathcal{A}$ is successful in its attack then at the end of the simulation there must exist a fresh instance of server S that has accepted without a partnered instance of client C. This means that $\mathcal{B}$ is successful whenever $\mathcal{A}$ is and thus $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{CAuth}_{BPR}}(\kappa) \leq \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. By assumption in the theorem the left part of this inequality is negligible. Thus, $\mathcal{P}$ achieves client authentication in the model from [7].

Server Authentication: In order to argue that $\mathcal{P}$ also achieves server authentication as defined in [7], we recall that for symmetric PAKE protocols $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{CAuth}_{BPR}}(\kappa) = \mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{SAuth}_{BPR}}(\kappa)$. Since our theorem assumes that $\mathcal{P}$ is a symmetric $(1, 0, 0)$-MFAKE protocol we can use the inequality for client authentication above to immediately obtain $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{SAuth}_{BPR}}(\kappa) \leq \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. □

Special Case of $(0, 1, 0)$-MFAKE Using $(\alpha, \beta, \gamma) = (0, 1, 0)$ we obtain the setting of Two-Party Authenticated Key Exchange (2-AKE), as defined in [8] and refined later for the public key setting in [10]. This relationship is established in Theorem 4.

Theorem 4. *(1) Let $\mathcal{P}$ be an AKE-secure 2-AKE protocol with mutual authentication according to the definitions by Bellare and Rogaway [8], refined for the public key setting by Blake-Wilson, Johnson, and Menezes [10]. Then $\mathcal{P}$ is also an AKE-secure $(0, 1, 0)$-MFAKE protocol with client and server authentication. (2) Let $\mathcal{P}$ be an AKE-secure $(0, 1, 0)$-MFAKE protocol with client and server authentication. Then $\mathcal{P}$ is also an AKE-secure 2-AKE protocol with mutual authentication as defined in [8,10] in the client server communication model.*

Proof. We denote the notions of AKE-security and mutual authentication from [8,10] as AKE_{BR} and MAuth_{BR} respectively. For corresponding definitions we refer to [8,10]. The two statements in the theorem are proven separately.

Statement (1) Let $\mathcal{A}^{\mathrm{AKE}}$ be an adversary against the AKE-security of $\mathcal{P}$, $\mathcal{A}^{\mathrm{SAuth}}$ an adversary against the SAuth-security of $\mathcal{P}$ and $\mathcal{A}^{\mathrm{CAuth}}$ an adversary against the CAuth-security of $\mathcal{P}$ in the $(1, 0, 0)$-MFAKE model. Using $\mathcal{A}^{\mathrm{AKE}}$ we will construct an adversary $\mathcal{B}^{\mathrm{AKE}_{BR}}$ against the AKE_{BR}-security of $\mathcal{P}$ and using any of $\mathcal{A}^{\mathrm{CAuth}}$ or $\mathcal{A}^{\mathrm{SAuth}}$ we will construct an adversary $\mathcal{B}^{\mathrm{MAuth}_{BR}}$ against the mutual authentication of $\mathcal{P}$ in the model from [8,10].

Simulation of Oracles: In order to simplify the notation we will use in this paragraph $\mathcal{A}$ for an adversary in our $(0, 1, 0)$-MFAKE model and $\mathcal{B}$ for the adversary (against the same security property as $\mathcal{A}$) in the model from [8,10]. The general idea is that $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine and simulates oracles that are available to $\mathcal{A}$ in the $(0, 1, 0)$-MFAKE model using the oracles that are available to $\mathcal{B}$ in the model from [8,10]. Note that queries of $\mathcal{A}$ to the $\mathsf{BioComp}$ oracle need not to be handled, as the theorem views $\mathcal{P}$ as a 2-AKE protocol that (by definition) does not use biometrics.

If $\mathcal{A}$ invokes a new session between a client C and a server S via an $\mathsf{Invoke}(C, S)$ query then $\mathcal{B}$ queries $\mathsf{Send}(C, S, s, \lambda)$, where λ is the empty string, with a previously unused s and returns its output to $\mathcal{A}$. Note that in [8,10], protocol sessions are invoked through this special Send query.

Queries of $\mathcal{A}$ to its Send, $\mathsf{RevealSK}$, $\mathsf{CorruptClient}$, and $\mathsf{CorruptServer}$ oracles are answered by $\mathcal{B}$ using its own oracles as follows: If $\mathcal{A}$ asks a $\mathsf{Send}([\mathrm{U}, s], m)$ query with $\mathrm{U} \in \{C, S\}$ then $\mathcal{B}$ queries $\mathsf{Send}(\mathrm{U}, \mathsf{pid}(\mathrm{U}), s, m)$ from [8] and returns its output to $\mathcal{A}$. If $\mathcal{A}$ asks a $\mathsf{RevealSK}([\mathrm{U}, s])$ query with $\mathrm{U} \in \{C, S\}$ then $\mathcal{B}$ returns the output of its own $\mathsf{Reveal}(\mathrm{U}, \mathsf{pid}(\mathrm{U}), s)$ query from [8]. If $\mathcal{A}$ asks a $\mathsf{CorruptServer}(S)$ then $\mathcal{B}$ returns the output of its own $\mathsf{Corrupt}(S)$ query. In the $(0, 1, 0)$-MFAKE setting the $\mathsf{CorruptClient}$ queries of $\mathcal{A}$ can only be of the form $\mathsf{CorruptClient}(C, 2, 1)$. Any such query of $\mathcal{A}$ is forwarded by $\mathcal{B}$ as its own $\mathsf{Corrupt}(C)$ query and its response is handed over to $\mathcal{A}$. (Note that $\mathsf{Corrupt}$ oracle was introduced to the model from [8] in [10] to address forward secrecy in the context of public key-based AKE protocols.)

In this way $\mathcal{B}$ can perfectly simulate Invoke, Send, $\mathsf{RevealSK}$, $\mathsf{CorruptClient}$, and $\mathsf{CorruptServer}$ queries of $\mathcal{A}$.

AKE-Security: We first focus on the AKE-security, i.e. assume $\mathcal{A} = \mathcal{A}^{\mathrm{AKE}}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{AKE}_{BR}}$. Here $\mathcal{B}$ needs to simulate the additional Test oracle that is available to $\mathcal{A}$ in the $(0, 1, 0)$-MFAKE model and that can be asked for any fresh server instances. Note that also $\mathcal{B}$ has access to a Test oracle in the model from [8]. The difference is – similar to the proof of Theorem 3 and the model from [7] – that [8,10] follow the so-called Find-then-Guess (FtG) approach, where only one Test query can be asked, whereas $(0, 1, 0)$-MFAKE follows the Real-Or-Random (ROR) approach that allows for multiple Test queries. We again use the AKE modeling result by Abdalla, Fouque, and Pointcheval [5, Lemma 2] as an intermediate step, i.e. we first consider $\mathcal{B} = \mathcal{B}^{\mathrm{AKE}_{BR}}_{RoR}$ within our reduction (that is as an AKE-adversary for the ROR version of [8]) and let $\mathcal{B}$ answer each $\mathsf{Test}([S, s])$ query of $\mathcal{A}$ by forwarding it as its own $\mathsf{Test}(S, s)$ query and returning the output back to $\mathcal{A}$. Once $\mathcal{A}$ outputs bit b', indicating the end of the AKE-security game from the $(0, 1, 0)$-MFAKE model, $\mathcal{B}$ forwards this bit as its own output in the AKE-security game from the ROR version of [8]. If $\mathcal{A}$ is successful, i.e. correctly outputs bit $b' = b$ used by the Test oracle, then so is $\mathcal{B}$. This is guaranteed by the definitions of freshness. Now by switching to $\mathcal{B} = \mathcal{B}^{\mathrm{AKE}_{BR}}_{FtG}$ and using the result from [5, Lemma 2] we obtain the resulting inequality $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{AKE}}(\kappa) \le q_{\mathsf{Test}} \cdot \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{AKE}_{BR}}(\kappa)$, where q_{Test} is the number of Test queries asked by the AKE adversary $\mathcal{A} = \mathcal{A}^{\mathrm{AKE}}$ in the $(0, 1, 0)$-MFAKE model.

Mutual Authentication: We proceed with the notion of mutual authentication, i.e. assuming either $\mathcal{A} = \mathcal{A}^{\mathrm{CAuth}}$ or $\mathcal{A} = \mathcal{A}^{\mathrm{SAuth}}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{MAuth}_{BR}}$. This notion is simpler since there are no further Test queries to consider. If $\mathcal{A}^{\mathrm{CAuth}}$ is successful in its attack then at the end of the simulation there must exist a fresh instance of server S that has accepted without a partnered instance of client C. If $\mathcal{A}^{\mathrm{SAuth}}$ is successful in its attack then at the end of the simulation there must exist a fresh instance of client C that has accepted without a partnered instance of server S. This means that in both cases $\mathcal{B}$ is successful whenever $\mathcal{A}$ is and $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{CAuth}}(\kappa) \le \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{MAuth}_{BR}}(\kappa)$ as well as $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{SAuth}}(\kappa) \le \mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{MAuth}_{BR}}(\kappa)$.

Statement (2) Let $\mathcal{A}^{\mathrm{AKE}_{BR}}$ be an adversary against the AKE_{BR}-security of $\mathcal{P}$, $\mathcal{A}^{\mathrm{MAuth}_{BR}}$ an adversary against mutual authentication of $\mathcal{P}$, and $\mathcal{A}^{\mathrm{SAuth}_{BR}}$ an adversary against server authentication of $\mathcal{P}$ in the model from [8,10]. We will use $\mathcal{A}^{\mathrm{AKE}_{BR}}$ to construct an AKE-adversary $\mathcal{B}^{\mathrm{AKE}}$ in the $(0, 1, 0)$-MFAKE model. We will use $\mathcal{A}^{\mathrm{MAuth}_{BR}}$ to construct an adversary $\mathcal{B}^{\mathrm{CAuth}}$ against client authentication in the $(0, 1, 0)$-MFAKE model.

Simulation of Oracles: In order to simplify the notation we will use in this paragraph the notation $\mathcal{A}$ for an adversary in the model from [8,10] and $\mathcal{B}$ for the corresponding adversary in our $(0, 1, 0)$-MFAKE model. As in Case (1), $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine and simulates oracles that are available to $\mathcal{A}$ in the model from [8,10] using the oracles from the $(0, 1, 0)$-MFAKE model. Note that since the $(0, 1, 0)$-MFAKE protocol $\mathcal{P}$ does not use biometrics $\mathcal{B}$ has no use for its $\mathsf{BioComp}$ oracle. We discuss now, how $\mathcal{B}$ answers queries of $\mathcal{A}$ to its oracles Send and Reveal from [8], refined with the additional oracle $\mathsf{Corrupt}$ from [10].

The Send oracle in [8,10] invokes new sessions on the fly when the communication begins. In contrast in our $(0, 1, 0)$-MFAKE model sessions are explicitly invoked using the Invoke oracle. Hence, if $\mathcal{A}$ queries $\mathsf{Send}(\mathrm{U}, \mathrm{U}', s, \lambda)$ from [8,10], where λ is the empty string, to start a session, $\mathcal{B}$ calls $\mathsf{Invoke}(\mathrm{U}, \mathrm{U}')$, returns its output to $\mathcal{A}$, and records the resulting instance as the s-th instance of U with server U' as its communication partner. Here, we implicitely assume, that the protocol follows the client-server communication model, i.e. one of the participants will always be a client and the other one a server. For regular queries of the form $\mathsf{Send}(\mathrm{U}, \mathrm{U}', s, m)$, $\mathcal{B}$ needs to make sure, that the specified receiver already exists. If it does not exist, $\mathcal{B}$ first calls $\mathsf{Invoke}(\mathrm{U}, \mathrm{U}')$, and records the resulting instance as the s-th instance of U with U' as its communication partner. Once $\mathcal{B}$ has ensured that the receiving instance exists, the query can be answered by $\mathcal{B}$ by forwarding it as its own Send query in the $(1,0,0)$-MFAKE model and its response is handed over to $\mathcal{A}$.

A $\mathsf{Reveal}(\mathrm{U}, s)$ query of $\mathcal{A}$ is answered by $\mathcal{B}$ through its own $\mathsf{RevealSK}([\mathrm{U}, s])$ query.

A $\mathsf{Corrupt}(\mathrm{U})$ query of $\mathcal{A}$ is answered by $\mathcal{B}$ depending on whether $\mathrm{U} = C$ or $\mathrm{U} = S$. If $\mathrm{U} = C$ the query is answered by $\mathcal{B}$ though its own $\mathsf{CorruptClient}(\mathrm{U}, 2, 1)$ query. Otherwise the query is answered by $\mathcal{B}$ through its own $\mathsf{CorruptServer}(\mathrm{U})$ query.

AKE-security: We first consider the notion of AKE-security, i.e. assuming that $\mathcal{A} = \mathcal{A}^{\mathrm{AKE}_{BR}}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{AKE}}$. In this case $\mathcal{B}$ must additionally simulate the Test oracle that is available to $\mathcal{A}$ in [8,10]. Note that also $\mathcal{B}$ has access to a Test oracle in the $(0,1,0)$-MFAKE model but there is a differences between the two models. The first difference (which was important for Statement (1) above) is that $\mathcal{B}$ may ask multiple Test queries, whereas $\mathcal{A}$ may issue only one Test query. This difference is not important for Statement (2), since $\mathcal{B}$ executes $\mathcal{A}$ as a subroutine and will thus ask only one Test query. The second difference, which is important for Case (2), is that $\mathcal{B}$ can query its Test oracle with respect to server instances only (recall that in the (α, β, γ)-MFAKE setting servers do not necessarily have secret keys), whereas for $\mathcal{A}$ in the 2-AKE setting from [8,10] there is no distinction between clients and servers and it can direct its Test query to any instance. We deal with this difference in the following way. From $\mathcal{A}$'s point of view, the protocol is completely symmetric and he has no way to distinguish between clients and servers. Therefore, if $\mathcal{A}$ asks a Test query for some pair of communication partners, it will pick the server with probability $\frac{1}{2}$. If $\mathcal{A}$ chose a server instance for its Test query, $\mathcal{B}$ can always forward the $\mathsf{Test}(S, s)$ query of $\mathcal{A}$ as its own $\mathsf{Test}([S, s])$ query where S is a server in the $(1,0,0)$-MFAKE model. Otherwise, $\mathcal{B}$ needs to abort, as it cannot answer the Test query.

Once $\mathcal{A}$ outputs bit b', indicating the end of the AKE-security game from [8,10], $\mathcal{B}$ forwards this bit as its own output in the AKE-security game from the $(0,1,0)$-MFAKE model. If $\mathcal{A}$ is successful, i.e. correctly outputs bit $b' = b$ used by the Test oracle, then so is $\mathcal{B}$. This is guaranteed by the definitions of freshness in both models. This gives us the desired inequality $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{AKE}_{BR}}(\kappa) \leq 2\mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{AKE}}(\kappa)$.

Mutual Authentication: We continue with the notion of mutual authentication, i.e. assuming $\mathcal{A} = \mathcal{A}^{\mathrm{MAuth}_{BR}}$ and $\mathcal{B} = \mathcal{B}^{\mathrm{CAuth}}$. In this case there are no additional oracles to take care of. If $\mathcal{A}$ is successful in its attack then at the end of the simulation there must exist a fresh instance of participant U that has accepted without a partnered instance of U$'$. This means that $\mathcal{B}$ is successful whenever $\mathcal{A}$ is, as long as $\mathrm{U} = C$. We again argue that because of the protocols symmetry, the probability of $\mathrm{U} = C$ is $\frac{1}{2}$. Thus $\mathsf{Adv}^{\mathcal{P},\mathcal{A}}_{\mathrm{MAuth}_{BR}}(\kappa) \leq 2\mathsf{Adv}^{\mathcal{P},\mathcal{B}}_{\mathrm{CAuth}}(\kappa)$. Since by assumption in the theorem the left part of this inequality is negligible we follow that $\mathcal{P}$ achieves mutual authentication in the model from [8,10]. □

Special Case of $(0,0,1)$-MFAKE If we set the parameters $(\alpha, \beta, \gamma) = (0,0,1)$ our definitions of AKE-security and client authentication result in the stand-alone notion of Biometric-based Authenticated Key Exchange (BAKE) *in the presence of liveness assumption.* Working with public biometric data, we emphasize the difference of this setting to existing BAKE notions such as [12,15] that model biometric data as *secret data.* The latter are usually constructed using secure sketches and robust fuzzy extractors [18,11] and have an initial step, in which both BAKE participants (client and server) derive some secret high-entropy shared key K. The actual authentication is then performed using K in a (symmetric) two-party AKE protocol session and is analyzed using the model from [8].

A stand-alone BAKE setting with public distribution of biometric data can also be obtained from the MFAKE model in [36], which we take as a basis for our (α, β, γ)-MFAKE. A by-product of a BAKE model with liveness assumption is that its security definitions are equally applicable to biometrics of high and low entropy — which is not the case if biometrics are regarded as secrets — since in order to make active use of a biometric the adversary must consult its BioComp oracle.

Simple $(0,0,1)$-MFAKE Protocol. As observed in [36], stand-alone BAKE protocols with public biometric data have not been proposed so far. The actual MFAKE construction in [36] doesn't admit a pure BAKE protocol since it binds biometric data to the password and it is not clear how to separate the both. In fact this link lead to an attack in [25]. We give now a simple construction of a stand-alone AKE-secure BAKE, termed $(0,0,1)$-MFAKE, whose explicit matching process checks that the Hamming distance of a candidate template W'_C and the reference template W_C (stored at S) remains below a threshold τ:

Let $(\mathbb{G}, g, q)$ be a cyclic group of sufficiently large prime order q. C and S first execute an unauthenticated Diffie-Hellman key exchange in $\mathbb{G}$ by exchanging g^x and g^y. Consider two hash functions $\mathcal{H}_1, \mathcal{H}_2 : \mathbb{G} \mapsto \{0,1\}^\kappa$. Let $W'_{C,i}$ resp. $W_{C,i}$ denote the ith bit of the corresponding template. For each bit i the client computes $h_i = \mathcal{H}_1(g^x, g^y, g^{xy}, W'_{C,i}, i)$ using its version of g^{xy} and sends the resulting set $\{h_i\}_i$ to S. S re-computes corresponding values using its version of g^{xy} and the reference template W_C, and accepts with $k_S = \mathcal{H}_2(g^x, g^y, g^{xy})$ if strictly less than τ hash values from $\{h_i\}_i$ do not match. It is not difficult to see that this protocol guarantees AKE-security for the session keys k_S if liveness assumption is in place, which essentially prevents the adversary from sending any h_i that was not computed beforehand through the BioComp oracle. The secrecy of the session key k_S follows then from the classical CDH assumption in the random oracle model.

Special Case of $(1,1,1)$-MFAKE By construction, our (α, β, γ)-MFAKE model implies the previous MFAKE model from [36]. Therefore, any MFAKE protocol with AKE- and CAuth-security from [36] satisfies corresponding definitions in our model with $(\alpha, \beta, \gamma) = (1,1,1)$ and vice versa. Our model further extends [36] with the additional requirement of server authentication (SAuth-security). Note that attacks from [26] against the protocol in [36] were partially based on the missing server authentication property.

Special Case of $(\alpha,0,0)$-MFAKE The scenario covered by this case is where a client shares α passwords with the remote server. This case has also been explored in the context of Multi-Factor Password Authenticated Key Exchange (MFPAKE) introduced in [38]. Without formally establishing the relationship to the model from [38], we observe that like [38] our definitions also aim at AKE-security with forward secrecy of the established session keys as long as one of the passwords remains uncorrupted.